UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

INGENIERÍA TÉCNICA DE TELECOMUNICACIÓN: TELEMÁTICA

PROYECTO FINAL DE CARRERA

DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DE AYUDA A LA GENERACIÓN DE INFORMES DE MERCADO A PARTIR DE INFORMACIÓN EN REDES SOCIALES

Autora: Ana María Coleto Torres

Tutor: Julio Villena Román

Octubre de 2015

Agradecimientos

Por fin ha llegado el día en el que presento mi Proyecto Fin de Carrera y es que pensaba que este día no iba a llegar nunca. El camino ha sido largo y duro pero ya se acaba.

Quiero dar las gracias especialmente a mis padres. Gracias por la educación y los valores que me habéis transmitido desde pequeña. Sois el mejor ejemplo a seguir. Gracias por vuestro apoyo, comprensión y paciencia. Sin vosotros esto no hubiese sido posible.

A toda mi familia porque siempre creísteis que lo conseguiría. Pero en especial a mi prima Ana, mi compañera de batallas, porque siempre estas cuando te necesito.

A mis compañeros de universidad, en especial a Álvaro, Carlos, Juanal, Marcela, Marta, Paola, Bea, Raquel, Sergio, Jesús, Patri, Adolfo, Sergio Ace y Guille porque me habéis hecho el camino más fácil. Empezasteis siendo simples compañeros y os habéis convertido en muy buenos amigos. Sin vosotros nada hubiese sido igual.

A mis amigas de siempre, Rocío, Sheila, Ana, Sole, Pili y Bea porque siempre me habéis apoyado y animado a continuar.

A los amigos que han ido apareciendo por el camino Isra, Patri, José, David, Dani, Vero, Miguel, Sergio, Keko y Ana porque vuestras palabras de ánimo siempre me ayudan.

A mis amigos y compañeros de trabajo Carol, Iker, Elena y Aitor y a mi jefa Bea porque habéis sufrido conmigo esta última etapa y me habéis ayudado en todo lo que habéis podido.

Y por último a mi tutor de proyecto Julio, por su paciencia, comprensión y dedicación.

Resumen

Vivimos en una era en la que las redes sociales se han convertido en una de las principales vías de comunicación interpersonal, dejando obsoletos otros medios de comunicación como correo electrónico, teléfono, foros, SMS, etc.

Una de las redes sociales con más éxito es Twitter, que es un servicio gratuito de *microblogging* que permite a sus usuarios publicar mensajes de texto, denominados tweets cuya longitud máxima es de 140 caracteres.

El formato que usa Twitter es realmente sencillo, tanto en su presentación como en su uso. Quizá el ser intuitivo y fácil de manejar es lo que atraiga a tantos usuarios que actualmente hacen uso de esta red social. La utilización que los usuarios hacen de Twitter va desde el uso meramente personal hasta el uso de ámbito empresarial.

Una de las principales funcionalidades de Twitter es que permite publicar mensajes expresando opiniones, hablando sobre algún tema de interés o haciendo referencia a alguna marca o producto. Este mensaje a la vez puede propagarse por la red mediante retweet (RT)[1] de otros usuarios.

Recogiendo los tweets que contengan las palabras claves de marcas o productos que nos interesan podremos analizar el texto y averiguar el sentimiento que tiene el usuario con respecto al tema. También podremos conocer diferentes conceptos o entidades utilizadas en el tweet, países desde donde se han publicado los tweets, temas en los que se clasifica el tweet, etc.

El principal objetivo de este proyecto es recoger la información de redes sociales analizarla y mostrar los resultados para elaborar informes de mercado en base a los datos recogidos.

Los informes de mercado son herramientas útiles para enfocar un negocio y examinar la viabilidad de un proyecto. Este tipo de estudios ayudan a entender a los clientes, a los competidores y al mercado en sí mismo. También puede ayudar al emprendedor a poder desarrollar un nuevo plan de negocio, lanzar nuevos productos y servicios y expandirse en un nuevo mercado.

Para alcanzar el objetivo propuesto este proyecto recoge los últimos tweets relacionados con una marca, los analiza mediante librerías de análisis del texto, y muestra una serie de resultados de forma visual.

[1] si un usuario hace un RT de un tweet de otro usuario de Twitter, es como si reenviase dicho tweet a todos sus seguidores

Abstract

We live in an era in which social networks have become one of the main ways of interpersonal communication, leaving obsolete other media such as email, phone, forums, SMS, etc.

One of the most successful social networks is Twitter, which is a free service of microblogging that allows users to publish text messages, called tweets, whose maximum length is 140 characters.

The format used by Twitter is really simple, both in its presentation and in its use. Perhaps being intuitive and easy to use is what attracts so many users that currently make use of this social network. The use that the users make Twitter goes from merely personal use to business use.

One of the main features of Twitter is that it allows to post messages expressing opinions, talking about a topic of interest, or referring to any brand or product. This message at the same time can be spread through the network via retweet (RT) by other users.

After collecting the tweets containing keywords of brands or products that interest us we can analyze the text and find out the feeling that the user has with respect to the subject. You can also discover different concepts or entities used in the tweet, countries of published tweets, topics which is classified the tweet, etc.

The main objective of this project is to collect analyze social networking information and show the results for reporting of market on the basis of the data collected.

Market reports are useful tools to focus on a business and examine the feasibility of a project. Such studies help to understand customers, competitors and the market itself. You can also help the entrepreneur to develop a new business plan, launching new products and services and expand into a new market.

To achieve the objective proposed this project collects the latest tweets related to a brand, analyzes them using text analysis libraries, and shows a number of results in a visual way.

Índice general

Índice de figuras

Índice de tablas

Lista de acrónimos

- **BSD -** *Berkeley Software Distribution*
- **CSRF -** *Cross Site Request Forgery*
- **CSS -** *Cascading Style Sheets*
- **HTML -** *HyperText Markup Languague*
- **MVC -** *Modelo Vista Controlador*
- **MVT -** *Modelo Vista Template*
- **OAuth -** *Open Authorization*
- **PC -** *Personal Computer*
- **REST -** *Representational State Transfer*
- **WSGI -** *Web Server Gateway Interface*
- **W3C -** *World Wide WebConsortium*

Capítulo 1

Motivación y contexto del proyecto

1.1. Motivación

Estamos viendo que en los últimos años la sociedad se ha modernizado notablemente. Se está registrando un gran cambio en las formas de comunicación interpersonales. Atrás quedaron las vías de comunicación como el teléfono, o el correo convencional. En la actualidad la gente elige las redes sociales, como Facebook[2] o Twitter[3], como vías de comunicación y también como vía de expresión con respecto a temas de actualidad o productos del mercado.

La llegada de nuevas tecnologías como los smartphones ha facilitado este movimiento. Hoy en día la mayoría de la población posee un smartphone con tarifa plana de Internet que le permite estar conectado a las redes sociales tanto dentro como fuera de casa. Es por eso que las redes sociales toman gran importancia ya que te permiten comunicarte y estar informado en tiempo real desde cualquier punto. Un buen ejemplo de una red que tiene gran popularidad es Twitter, en la que está basada este proyecto.

Twitter [1] es un servicio de *microblogging* que surgió en el año 2006, creado por Jack Dorsey. La simplicidad de su diseño y la facilidad de uso lo hacen muy atractivo al usuario, quizás en ello radique su éxito mundial. Se estima que tiene más de 200 millones de usuarios y que genera más de 65 millones de tweets y más de 800000 peticiones de búsqueda diariamente. Estos datos han hecho que se le conozca como el SMS de Internet.

Esta red social permite publicar mensajes de texto, llamados tweets, con una longitud máxima de 140 caracteres.

Aquel usuario que tenga una cuenta abierta en Twitter podrá seguir a otros usuarios; puede ser seguido; puede enviar mensajes directos (DMs), que son mensajes privados dirigidos a un usuario en concreto; y puede hacer retweets (RTs). Los tweets pueden contener menciones, o lo que es lo mismo, estar dirigidos a otros usuarios en concreto; y también pueden contener hashtags, que son palabras precedidas por el carácter '#' que sirven para etiquetar los tweets y facilitar las búsquedas por temáticas de los mismos.

Como se ha dicho anteriormente, son numerosos los posibles usos de Twitter: empresarial, personal, informativo, etc. Es por estos usos por lo que surge la idea de este proyecto.

[2] https://www.facebook.com/
[3] https://twitter.com/?lang=es

La realidad es que un usuario de Twitter hace uso de esta aplicación donde publica tanto información personal, como cualquier tipo de opinión sobre diferentes temas, etc.

Este proyecto surge con la idea de recoger esas opiniones de los usuarios para hacer una comparación sobre distintas marcas contemplando varios aspectos.

La finalidad es saber la opinión de los usuarios, para saber qué direcciones futuras deberá tomar la empresa objeto del estudio.

1.2. Objetivos

El objetivo principal de este proyecto fin de carrera es el diseño e implementación de una aplicación web para que genere un informe de mercado a través de información recogida en Twitter.

El usuario de la aplicación deberá rellenar una serie de campos para autenticarse en Twitter y en MeaningCloud, el servicio web de análisis de texto que se ha seleccionado, además deberá introducir las marcas que quiere consultar y el número de resultados que desea mostrar. Estas marcas se pasarán como parámetros en las consultas a Twitter. Una vez obtenidos los tweets la aplicación elaborará un análisis consultando a las librerías de MeaningCloud y mostrará los resultados de los diferentes análisis por pantalla. Los resultados se mostrarán en forma de gráficos para que la comparación sea más intuitiva para el usuario, a simple vista.

Los tipos de gráficos que se muestran a la salida del programa son: gráficos de sectores, gráficos de barras y gráficos de texto.

1.3. Estructura de la documentación

La memoria se compone de siete capítulos y un apéndice. A lo largo de todos estos capítulos se exponen detalladamente los aspectos considerados relevantes para el desarrollo del proyecto.

Se pasa a detallar brevemente que contiene cada uno de los capítulos de esta memoria.

- **Capítulo1. Motivación y contexto del proyecto.**
  En este capítulo se detallan los objetivos propuestos para el desarrollo del proyecto.

- **Capítulo 2. Estado del arte.**
  Capitulo en el que se detallan los aspectos más técnicos del proyecto. Se hace un resumen de las distintas tecnologías usadas para la elaboración del proyecto, lenguaje de programación Python, framework Django, librería Matplotlib, hojas de estilos CSS. Y una introducción a los estudios de mercado.

- **Capítulo 3. Requisitos de diseño.**
  Detalle del funcionamiento definido para la aplicación y criterios de diseño que se han seguido.

- **Capítulo 4. Diseño de alto nivel.**
  Como el título indica se hablará del diseño a alto nivel de la aplicación. También se hablará del uso que se le puede dar a la aplicación cumpliendo con los requisitos de la misma.

- **Capítulo 5. Implementación.**
  En este capítulo se detalla cómo se ha llevado a cabo la implementación de la aplicación. Diseño de la misma, estructura de ficheros, capas de aplicación, etc.

- **Capítulo 6. Pruebas.**
  En este capítulo se detallan cada una de las pruebas realizadas para comprobar que la aplicación desarrollada cumple la funcionalidad que se requería al inicio del proyecto.

- **Capítulo 7. Conclusiones y trabajos futuros.**
  Se describen las conclusiones a las que se ha llevado durante la realización del proyecto. Detalle de los obstáculos que han ido surgiendo por el camino y posibles mejoras de la aplicación para un futuro.

- **Apéndice A. Planificación.**
  En este apéndice se describe la historia del proyecto. La planificación, distribución temporal, costes, etc.

Capítulo 2

Estado del arte

En este capítulo se detallan las distintas tecnologías y herramientas utilizadas para la realización del proyecto.

Lo primero que se detalla en este capítulo es el lenguaje de programación utilizado, Python, así como el framework, Django. También se habla de la biblioteca Matplotlib que es la encargada de pintar los resultados tras las consultas y el análisis. Estas herramientas han sido utilizadas para elaborar el código fuente de la aplicación.

Se habla de la API REST de Twitter, y de las APIs de MeaningCloud utilizadas para la obtención de los tweets y su posterior análisis.

También se habla de CSS, utilizado para definir la presentación en pantalla de cada una de las plantillas de la aplicación.

Y por último se habla de los estudios de mercado.

2.1. Python

Se ha elegido este lenguaje de programación porque es fácil de aprender y el código es fácilmente legible. Además este lenguaje posee numerosas librerías con distintas funcionalidades, entre ellas la librería Matplotlib, que ha sido la elegida para la representación de los resultados.

Otro de los motivos por los que se ha elegido este lenguaje es por Django, un framework de Python muy potente y que tiene un servidor local de pruebas muy útil.

2.1.1. Introducción

Python [2] [3] [4] es un lenguaje de programación interpretado, usa típado dinámico y es multiplataforma. Su máxima filosofía es que el código sea legible.

Se trata de un lenguaje de programación multiparadigma, porque soporta orientación a objetos, programación imperativa y programación funcional, aunque en menor medida que las anteriores.

Este lenguaje es administrado por la *Python Software Foundation* y posee una licencia de código abierto, denominada *Python Software Foundation License*.

Python fue creado a finales de los años ochenta por Guido van Rossum, quien le puso ese nombre por su gran afición por los humoristas británicos Monty Python. Este lenguaje fue creado como sucesor del lenguaje de programación ABC, capaz de manejar excepciones e interactuar con el sistema operativo Amoeba. El lugar de creación de este lenguaje es el centro para las Matemáticas y la Informática (CWI, Centrum Wiskunde & Informática), en los Países Bajos.

Van Rossum es el principal autor de Python y el encargado en decidir la dirección de Python.

Actualmente existen dos versiones activas de Python:

- Python 2 es la más compatible ya que como ha estado más tiempo en el mercado hay gran número de librerías que pueden ser usadas con esta versión.
- Python 3 presenta funcionalidades mejoradas con respecto a la versión anterior pero prácticamente incompatible con Python 2.

2.1.2. Características

Python es un lenguaje simple y minimalista. Leer un buen programa de Python es como leer ingles. El pseudocódigo natural de Python es una de sus grandes virtudes ya que permite concentrarse en la solución del problema en lugar de la sintaxis.

Con Python es muy sencillo iniciarse en la programación ya que la sintaxis que ofrece es muy simple.

Los usuarios de Python se refieren a menudo a la **Filosofía Python** que es bastante análoga a la filosofía de Unix. Estos principios fueron famosamente descritos por el desarrollador de Python Tim Peters en *El Zen de Python*:

- Bello es mejor que feo.
- Explícito es mejor que implícito.
- Simple es mejor que complejo.
- Complejo es mejor que complicado.
- Plano es mejor que anidado.
- Disperso es mejor que denso.
- La legibilidad cuenta.
- Los casos especiales no son tan especiales como para quebrantar las reglas.
- Lo práctico gana a lo puro.
- Los errores nunca deberían dejarse pasar silenciosamente.
- A menos que hayan sido silenciados explícitamente.
- Frente a la ambigüedad, rechaza la tentación de adivinar.
- Debería haber una -y preferiblemente sólo una- manera obvia de hacerlo.
- Aunque esa manera puede no ser obvia al principio a menos que usted sea holandés.
- Ahora es mejor que nunca.
- Aunque nunca es a menudo mejor que ya mismo.
- Si la implementación es difícil de explicar, es una mala idea.
- Si la implementación es fácil de explicar, puede que sea una buena idea.

- Los espacios de nombres (namespaces) son una gran idea ¡Hagamos más de esas cosas!

Python es un ejemplo de un FLOSS (Free/Libre and Open Source Software - Gratuito/Libre y Software de Fuente Abierta). Se puede distribuir libremente copias de este software, leer su código fuente, hacerle cambios, usar partes del mismo en nuevos programas libres, y en general lo que se quiera. FLOSS está basado en un concepto de una comunidad que comparte conocimiento. Esta es una de las razones por las cuales Python es tan bueno, ha sido creado y mejorado por una comunidad que solo quiere ver un mejor Python.

Con Python nunca hay que preocuparse por detalles de bajo nivel como por ejemplo, manejar la memoria empleada en el programa.

Debido a su naturaleza de ser Open Source, Python ha sido portado a diversas plataformas. Se puede usar Python sobre Linux, Windows, Macintosh, Solaris, OS/2, Amiga, AROS, AS/400, BeOS, OS/390, z/OS, Palm OS, QNX, VMS, Psion, Acorn RISC OS, VxWorks, PlayStation, Sharp Zaurus, Windows CE y PocketPC.

Con un lenguaje interpretado, como Python, no existen compilaciones separadas y pasos de ejecución. Solo se ejecuta el programa desde el código fuente. Internamente, Python convierte el código fuente en una forma intermedia llamada bytecodes, después los traduce en el lenguaje nativo de tu computadora y ejecuta. Todo esto hace el uso de Python mucho más sencillo. Solo debes *ejecutar* tus programas, no debes preocuparte sobre enlazar y cargar librerías, etc. Esto lo convierte en portable, ya que solo debes copiar el código de tu programa Python en cualquier otro sistema y trabajará igualmente.

Python permite programación orientada a procedimientos así como orientada a objetos. En lenguajes *orientados a procedimientos*, el programa está construido sobre procedimientos o funciones los cuales no son nada más que piezas de programa reutilizables. En lenguajes *orientados a objetos*, el programa es construido sobre objetos los cuales combinan datos y funcionalidad. Python ofrece una manera muy potente y simple de emplear programación orientada a objetos, especialmente, cuando se compara con lenguajes como C++ o Java.

Si necesitas que una pieza de código se ejecute muy rápido, puedes lograrlo escribiéndola en C y después combinarla con tu programa de Python.

Puedes insertar Python dentro de tu programa en C/C++ para ofrecer las facilidades de "scripting" dentro del mismo.

La librería estándar de Python es muy amplia. Puede ayudarte a hacer varias cosas que involucran: expresiones regulares, generación de documentos, evaluación de unidades, pruebas, procesos, bases de datos, navegadores web, CGI, ftp, correo electrónico, XML, XML-RPC, HTML, archivos WAV, criptografía, GUI (Graphical User Interfaces/interfaz gráfica de usuario) usando Tk, y también otras funciones dependientes del sistema. Todo esto está siempre disponible en cualquier sitio donde Python se instala y forma parte de la denominada filosofía de Python "batteries included" ("baterías incluidas").

Además de la librería estándar, hay otras librerías de calidad superior como el Python Imaging Library que es una sorprendente librería para la manipulación de imágenes.

2.1.3. Instalación

Para instalar Python, se debe descargar de la página oficial (https://www.python.org/downloads/) la versión que deseemos utilizar. En este caso se ha descargado la versión Python 2.7.9 para Windows, ya que como he mencionado en apartados anteriores es más compatible con distintas librerías.

- Ejecutar el instalador. Se abre un programa de instalación, se selecciona la opción "Instalar para todos los usuarios" y luego se pincha en "Siguiente".

- Hay que escoger un directorio de instalación para Python. Se recomienda el valor por defecto, que en este caso es Python27.

- Elegir los componentes que se desee instalar, en este caso se ha seleccionado la opción "Add Python.exe to Path". Después hacer clic en "Siguiente" para iniciar la instalación. Esperar unos minutos hasta que se complete el proceso de instalación y pinchar en la opción "Finalizar".

- Hacer clic en "Programas", "Python2.7" y luego "Python" desde el menú Inicio de Windows para probarlo. Una ventana se abrirá con un comando interactivo de Python del sistema. Una vez se haya confirmado que el programa está instalado correctamente, se cierra la ventana.

- Abrir un prompt y cambiar al directorio de Python. Se escribe "python" y se presiona "Enter" para iniciar la línea de comandos de Python del sistema.

2.1.4. Ejecución de código Python

Existen dos formas de ejecutar código Python:

1. **Sesión interactiva**

   Se escriben las líneas de código en el intérprete y se obtiene una respuesta del intérprete para cada línea.

   Para entrar en el intérprete de Python hay que teclear la siguiente sentencia en la consola de comandos:

```
> python
```

   El *prompt* cambiará y se podrá escribir directamente el código Python que se desea ejecutar. Por ejemplo, una vez dentro del intérprete, si se escribe la línea de código `print "Hola Mundo"`, como resultado de la ejecución aparecerá `Hola Mundo`:

```
>>> print "Hola Mundo"
```

```
Hola Mundo
```

2. **Scripts**

   Con este modo, tenemos las líneas del código escritas en un archivo con extensión `.py`.

   Siguiendo el ejemplo anterior tendríamos el código en un fichero de texto con el nombre de `hola.py` y ejecutaríamos el script de la siguiente manera:

```
> python hola.py

Hola Mundo
```

2.1.5. Comentarios

Podemos añadir comentarios a nuestro fichero Python de dos maneras. La primera y más utilizada para comentarios largos es `''' comentario '''`, y la segunda forma es añadiendo el símbolo #, este comentario es hasta final de línea.

El intérprete de Python no tiene en cuenta los comentarios.

```
'''
Comentario más largo en una línea en Python
'''
print "Hola mundo" # También es posible añadir un comentario al final de una línea de código
```

2.1.6. Variables

En Python las variables se definen de forma dinámica. Al declarar la variable no se especifica el tipo de dato y puede tomar distintos valores a lo largo del código.

```
x = 1
x = "texto" # Esto es posible porque los tipos son asignados dinámicamente
```

2.1.7. Tipos de datos

La siguiente tabla muestra un resumen de los tipos de datos:

Tipo	Clase	Notas	Ejemplo
`str`	Cadena	Inmutable	`'Cadena'`
`unicode`	Cadena	Versión Unicode de `str`	`u'Cadena'`
`list`	Secuencia	Mutable, puede contener objetos de diversos tipos	`[4.0, 'Cadena', True]`
`tuple`	Secuencia	Inmutable, puede contener objetos de diversos tipos	`(4.0, 'Cadena', True)`
`set`	Conjunto	Mutable, sin orden, no contiene duplicados	`set([4.0, 'Cadena', True])`
`frozenset`	Conjunto	Inmutable, sin orden, no contiene duplicados	`frozenset([4.0, 'Cadena', True])`
`dict`	Mapping	Grupo de pares clave:valor	`{'key1': 1.0, 'key2': False}`
`int`	Número entero	Precisión fija, convertido en *long* en caso de overflow.	`42`
`long`	Número entero	Precisión arbitraria	`42L` ó `456966786151987643L`
`float`	Número decimal	Coma flotante de doble precisión	`3.1415927`
`complex`	Número complejo	Parte real y parte imaginaria *j*.	`(4.5 + 3j)`
`bool`	Booleano	Valor booleano verdadero o falso	`True` o `False`

Figura 2. 1: Tipos de datos de Python [2]

Mutable: Si el valor puede cambiarse en tiempo de ejecución.

Inmutable: Si el valor no puede cambiar en tiempo de ejecución.

2.1.8. Listas y Tuplas

Listas

Para declarar una lista se usan los corchetes `'[]'` y los elementos dentro de esa lista van separados por `','`.

Pueden contener elementos de distintos tipos aunque suelen usarse para elementos del mismo tipo pero cantidad de elementos variable.

Para acceder a los elementos de una lista se utiliza un índice entero, comenzando por "0". También se pueden utilizar índices negativos para empezar por el final de la lista.

Las listas se caracterizan por ser mutables, es decir, se pueden modificar en tiempo de ejecución.

```
>>> lista = ["abc", 42, 3.1415]
>>> lista[0] # Acceder a un elemento por su índice
'abc'
>>> lista[-1] # Acceder a un elemento usando un índice negativo
3.1415
>>> lista.append(True) # Añadir un elemento al final de la lista
>>> lista
['abc', 42, 3.1415, True]
>>> del lista[3] # Borra un elemento de la lista usando un
índice (en este caso: True)
```

```
>>> lista[0] = "xyz" # Re-asignar el valor del primer elemento de la lista
>>> lista[0:2] # Mostrar los elementos de la lista del índice "0" al "2" (sin incluir este último)
['xyz', 42]
>>> lista_anidada = [lista, [True, 42L]] # Es posible anidar listas
>>> lista_anidada
[['xyz', 42, 3.1415], [True, 42L]]
>>> lista_anidada[1][0] # Acceder a un elemento de una lista dentro de otra lista (del segundo elemento, mostrar el primer elemento)
True
```

Tuplas

Para declarar una tupla se usan los paréntesis `'()'` y los elementos dentro de esa tupla van separados por `','`.

Pueden contener elementos de distintos tipos y suelen usarse para elementos de distintos tipos pero cantidad de elementos fija.

Para acceder a los elementos de una tupla se utiliza un índice entero, comenzando por "0". También se pueden utilizar índices negativos para empezar por el final de la tupla.

Las tuplas se caracterizan por ser inmutables, es decir, no se pueden modificar una vez creada.

```
>>> tupla = ("abc", 42, 3.1415)
>>> tupla[0] # Acceder a un elemento por su índice
'abc'
>>> del tupla[0] # No es posible borrar (ni añadir) un elemento en una tupla, lo que provocará una excepción
( Excepción )
>>> tupla[0] = "xyz" # Tampoco es posible re-asignar el valor de un elemento en una tupla, lo que también provocará una excepción
( Excepción )
>>> tupla[0:2] # Mostrar los elementos de la tupla del índice "0" al "2" (sin incluir este último)
('abc', 42)
>>> tupla_anidada = (tupla, (True, 3.1415)) # También es posible anidar tuplas
>>> 1, 2, 3, "abc" # Esto también es una tupla, aunque es recomendable ponerla entre paréntesis (recuerda que requiere, al menos, una coma)
(1, 2, 3, 'abc')
>>> (1) # Aunque entre paréntesis, esto no es una tupla, ya que no posee al menos una coma, por lo que únicamente aparecerá el valor
1
>>> (1,) # En cambio, en este otro caso, sí es una tupla
(1,)
>>> (1, 2) # Con más de un elemento no es necesaria la coma final
```

```
(1, 2)
>>> (1, 2,) # Aunque agregarla no modifica el resultado
(1, 2)
```

2.1.9. Diccionarios

Para declarar un diccionario se usan las llaves `'{}'`. Contienen elementos separados por comas, donde cada elemento está formado por un par `clave:valor`.

Los diccionarios son mutables, se puede cambiar el contenido de su `valor` en tiempo de ejecución, en cambio las `claves` son inmutables.

El `valor` asociado a la `clave` puede ser cualquier tipo de dato, incluso un diccionario.

```
>>> diccionario = {"cadena": "abc", "numero": 42, "lista":
[True, 42L]} # Diccionario que tiene diferentes valores por cada
clave, incluso una lista
>>> diccionario["cadena"] # Usando una clave, se accede a su
valor
'abc'
>>> diccionario["lista"][0] # Acceder a un elemento de una lista
dentro de un valor (del valor de la clave "lista", mostrar el
primer elemento)
True
>>> diccionario["cadena"] = "xyz" # Re-asignar el valor de una
clave
>>> diccionario["cadena"]
'xyz'
>>> diccionario["decimal"] = 3.1415927 # Insertar un nuevo
elemento clave:valor
>>> diccionario["decimal"]
3.1415927
>>> diccionario_mixto = {"tupla": (True, 3.1415), "diccionario":
diccionario} # También es posible que un valor sea un
diccionario
>>> diccionario_mixto["diccionario"]["lista"][1] # Acceder a un
elemento dentro de una lista, que se encuentra dentro de un
diccionario
42L
>>> diccionario = {("abc",): 42} # Sí es posible que una clave
sea una tupla, pues es inmutable
>>> diccionario = {["abc"]: 42} # No es posible que una clave
sea una lista, pues es mutable, lo que provocará una excepción
( Excepción )
```

2.1.10. Conjuntos

Los conjuntos se construyen mediante `set(items)`. Ítems puede ser cualquier objeto iterable como listas o tuplas. Los conjuntos no mantienen el orden ni contienen elementos duplicados.

Se suelen utilizar para eliminar duplicados de una secuencia, o para operaciones matemáticas como intersección, unión, diferencia y diferencia simétrica.

```
>>> conjunto_inmutable = frozenset(["a", "b", "a"]) # Se utiliza una lista como objeto iterable
>>> conjunto_inmutable
frozenset(['a', 'b'])
>>> conjunto1 = set(["a", "b", "a"]) # Primer conjunto mutable
>>> conjunto1
set(['a', 'b'])
>>> conjunto2 = set(["a", "b", "c", "d"]) # Segundo conjunto mutable
>>> conjunto2
set(['a', 'c', 'b', 'd']) # Recuerda, no mantienen el orden, como los diccionarios
>>> conjunto1 & conjunto2 # Intersección
set(['a', 'b'])
>>> conjunto1 | conjunto2 # Unión
set(['a', 'c', 'b', 'd'])
>>> conjunto1 - conjunto2 # Diferencia (1)
set([])
>>> conjunto2 - conjunto1 # Diferencia (2)
set(['c', 'd'])
>>> conjunto1 ^ conjunto2 # Diferencia simétrica
set(['c', 'd'])
```

2.1.11. Listas por compresión

Una lista por compresión es una expresión compacta para definir listas.

```
>>> range(5) # La función "range" devuelve una lista, empezando en 0 y terminando con el número indicado menos uno
[0, 1, 2, 3, 4]
>>> [i*i for i in range(5)] # Por cada elemento del rango, lo multiplica por sí mismo y lo agrega al resultado
[0, 1, 4, 9, 16]
>>> lista = [(i, i + 2) for i in range(5)]
>>> lista
[(0, 2), (1, 3), (2, 4), (3, 5), (4, 6)]
```

2.1.12. Funciones

Las funciones se definen con la palabra clave `def`, seguida por el nombre de la función y sus parámetros.

El valor devuelto en las funciones con `def` será devuelto con la instrucción `return`.

```
>>> def suma(x, y = 2):
     return x + y # Retornar la suma del valor de la variable "x" y el valor de "y"
>>> suma(4) # La variable "y" no se modifica, siendo su valor: 2
6
```

```
>>> suma(4, 10) # La variable "y" sí se modifica, siendo su nuevo valor: 10
14
```

2.1.13. Clases

Las clases se definen con la palabra clave `class` seguida del nombre de la clase y si hereda de otra clase, el nombre de esta.

En una clase un "método" equivale a una "función", y un "atributo" equivale a una "variable".

El método especial "`__init__`" se ejecuta al instanciar la clase. Se usa para inicializar atributos y ejecutar métodos necesarios.

Los atributos que se desee que sean accesibles desde fuera de la clase se deben declarar usando `self` delante del nombre.

En Python no existe el concepto de encapsulación, el programador debe asignar los valores a los atributos.

```
>>> class Persona(object):
    def __init__(self, nombre, edad):
        self.nombre = nombre # Un atributo cualquiera
        self.edad = edad # Otro atributo cualquiera
    def mostrar_edad(self): # Es necesario que, al menos, tenga un parámetro, generalmente: "self"
        print self.edad # mostrando un atributo
    def modificar_edad(self, edad): # Modificando Edad
        if edad < 0 or edad > 150: # Se comprueba que la edad no sea menor de 0 (algo imposible), ni mayor de 150 (algo realmente difícil)
            return False
        else: # Si está en el rango 0-150, entonces se modifica la variable
            self.edad = edad # Se modifica la edad

>>> p = Persona("Alicia", 20) # Instanciamos la clase, como se puede ver, no se especifica el valor de "self"
>>> p.nombre # La variable "nombre" del objeto sí es accesible desde fuera
'Alicia'
>>> p.nombre = "Andrea" # Y por tanto, se puede cambiar su contenido
>>> p.nombre
'Andrea'
>>> p.mostrar_edad() # Podemos llamar a un método de la clase
20
>>> p.modificar_edad(21) # Y podemos cambiar la edad usando el método específico que hemos hecho para hacerlo de forma controlada
>>> p.mostrar_edad()
21
```

2.1.14. Condicionales

Una sentencia condicional (if) ejecuta su bloque de código interno sólo si se cumple la condición. Se define usando la palabra clave `if` seguida de la condición y el bloque de código. Si hay condiciones adicionales se introducen usando `else if` o `elif` seguidas de la condición y su bloque de código. Puede haber un bloque final con la palabra clave `else` seguida de un bloque de código.

El código se ejecuta secuencialmente hasta que se encuentra una condición verdadera y se ejecuta sólo ese bloque de código. Si todas las condiciones fueran falsas se ejecuta el código correspondiente al bloque `else` (en caso de que exista).

```
>>> verdadero = True
>>> if verdadero: # No es necesario poner "verdadero == True"
      print "Verdadero"
    else:
      print "Falso"

Verdadero
>>> lenguaje = "Python"
>>> if lenguaje == "C": # lenguaje no es "C", por lo que este
bloque se obviará y evaluará la siguiente condición
      print "Lenguaje de programación: C"
   elif lenguaje == "Python": # Se pueden añadir tantos bloques
"elif" como se quiera
      print "Lenguaje de programación: Python"
   else: # En caso de que ninguna de las anteriores condiciones
fuera cierta, se ejecutaría este bloque
      print "Lenguaje de programación: indefinido"

Lenguaje de programación: Python
>>> if verdadero and lenguaje == "Python": # Uso de "and" para
comprobar que ambas condiciones son verdaderas
      print "Verdadero y Lenguaje de programación: Python"

Verdadero y Lenguaje de programación: Python
```

2.1.15. Bucle for

Recorre un objeto iterable como una lista, tupla, etc. Por cada elemento del iterable ejecuta un bloque de código.

Se define con la palabra clave `for` seguida de un nombre de variable, seguido de `in`, seguido del iterable, y por último el bloque de código. En cada iteración, el elemento siguiente del iterable se asigna al nombre de variable especificado.

```
>>> lista = ["a", "b", "c"]
>>> for i in lista: # Iteramos sobre una lista, que es iterable
      print i

a
b
```

```
c
>>> cadena = "abcdef"
>>> for i in cadena: # Iteramos sobre una cadena, que también es iterable
      print i, # Añadiendo una coma al final hacemos que no introduzca un salto de línea, sino un espacio

a b c d e f
```

2.1.16. Bucle while

El bucle while evalúa una condición y, si es verdadera, ejecuta el bloque de código. Continua evaluando y ejecutando mientras la condición sea verdadera.

Se define con la palabra clave `while` seguida de la condición, y a continuación el bloque de código.

```
>>> numero = 0
>>> while numero < 10:
      numero += 1
      print numero,

1 2 3 4 5 6 7 8 9
```

2.1.17. Módulos

Hay muchas propiedades que se pueden agregar al lenguaje importando módulos, que son "minicódigos" que proveen de ciertas funciones y clases para realizar determinadas tareas.

Los módulos se agregan a los códigos escribiendo `import` seguida del nombre del módulo que se quiera usar.

```
>>> import os # Módulo que provee funciones del sistema operativo
>>> os.name # Devuelve el nombre del sistema operativo
'windows'
>>> os.mkdir("/tmp/ejemplo") # Crea un directorio en la ruta especificada
>>> import time # Módulo para trabajar con fechas y horas
>>> time.strftime("%Y-%m-%d %H:%M:%S") # Dándole un cierto formato, devuelve la fecha y/o hora actual
'2015-09-125 18:01:17'
```

2.2. Django

2.2.1. Introducción

Django [5] es un framework de desarrollo web de código abierto. Está escrito en lenguaje Python, y respeta el patrón de diseño conocido como *Modelo-vista-controlador*.

Originalmente este framework fue desarrollado para gestionar varias páginas orientadas a noticias de la World Company de Lawrence, Kansas, y fue liberada al público bajo una licencia BSD en julio de 2005. El nombre fue elegido en honor al guitarrista de jazz gitano Django Reinhardt.

A partir de junio de 2008 es Django Software Foundation quien pasa a encargarse de Django.

El objetivo principal de Django es facilitar la creación de sitios web complejos. Pone especial énfasis en la reutilización, conectividad y extensibilidad de componentes, el desarrollo rápido y el principio DRY[4].

Python es usado en todas las partes del framework, incluidas configuraciones, archivos y en los modelos de datos.

Django requiere Python 2.5 o superior. Además en su entorno de desarrollo Django tiene un servidor propio ligero para hacer pruebas, con la restricción de que sólo permite un usuario a la vez

2.2.2. Características

Django se usó un tiempo en producción antes de que fuese liberado al público. Fue desarrollado por Adrian Holovaty, Simon Willison, Jacob Kaplan-Moss Miner mientras trabajaban en World Online. En sus comienzos se utilizó para administrar tres sitios web de noticias: The Lawrence Journal-World, lawrence.com y KUsports.com

Django facilita una serie de características que hace muy sencillo el desarrollo rápido de páginas orientadas a contenidos. Un ejemplo, es que en lugar de requerir que los desarrolladores escriban controladores y vistas para las áreas de administración de la página, Django proporciona una aplicación incorporada para administrar los contenidos, que puede incluirse como parte de cualquier página hecha con Django y que puede administrar varias páginas hechas con Django a partir de una misma instalación.

La aplicación administrativa permite la creación, actualización y eliminación de objetos de contenido, llevando un registro de todas las acciones realizadas sobre cada uno, y proporciona una interfaz para administrar los usuarios y los grupos de usuarios.

La distribución principal de Django también agrupa aplicaciones que proporcionan un sistema de comentarios, "páginas planas" que permiten gestionar páginas de contenido

[4] Del inglés *Don't Repeat Yourself*

sin necesidad de escribir controladores o vistas para esas páginas, y un sistema de redirección de URLs.

Otras características:

- Un mapeador objeto-relacional. Es un modelo de programación que consiste en transformar una tabla de la base de datos en una entidad que simplifica las tareas básicas de acceso, modificación y eliminación de los datos.
- Aplicaciones que pueden instalarse en cualquier página gestionada con Django.
- Una robusta API de bases de datos.
- Un sistema incorporado de "vistas genéricas" que ahorra tener que escribir la lógica de ciertas tareas comunes.
- Un sistema extensible de plantillas basado en etiquetas, con herencia de plantillas.
- Un despachador de URLs basado en expresiones regulares.
- Un sistema "middleware" para desarrollar características adicionales; por ejemplo, la distribución principal de Django incluye componentes middleware que proporcionan cacheo, compresión de la salida, normalización de URLs. Protección CSRF y soporte de sesiones.
- Soporte de internacionalización, incluyendo traducciones incorporadas de la interfaz de administración.
- Documentación incorporada accesible a través de la aplicación administrativa. Esto incluye documentación generada automáticamente de los modelos y las bibliotecas de plantillas añadidas por las aplicaciones.

2.2.3. Arquitectura

Django está inspirado en la filosofía de desarrollo *Modelo Vista Controlador*, aunque sus desarrolladores declaran públicamente que no se sienten especialmente atados a observar estrictamente ningún paradigma particular, y prefieren hacer "lo que les parece correcto". El resultado es que la arquitectura de Django es una variación de MVC[5] llamada Modelo Vista Template (MVT).

Para entender mejor el funcionamiento de MTV:

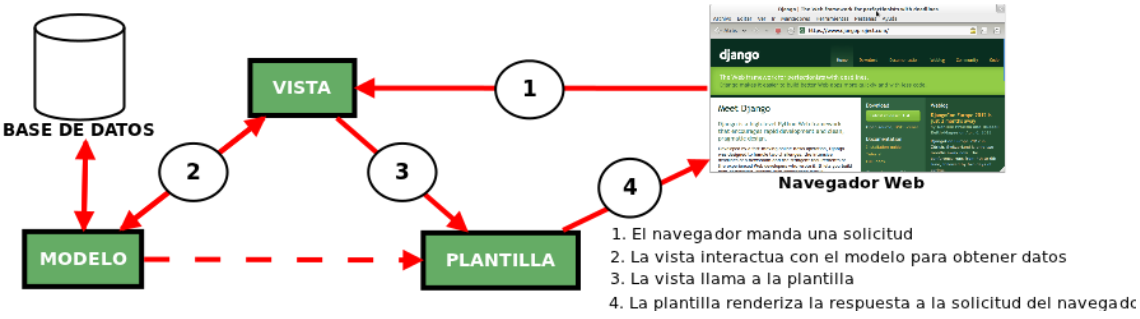


Figura 2. 2: Funcionamiento de Django

[5] Modelo Vista Controlador

En la figura anterior se observa como el navegador envía una solicitud a la Vista, esta llama al Modelo para obtener los datos de la correspondiente base de datos y posteriormente se los devuelve a la Vista. Una vez que la Vista tiene los datos se los envía a la Plantilla y está redirige la respuesta al navegador.

Django permite que los desarrolladores se dediquen a construir los objetos Entity y la lógica de presentación, gracias al poder de las capas *mediator* y *foundation.*

Django presenta las siguientes capas:

Presentación

En Django la interacción entre el usuario y el computador la realizan el *template engine* y el *template loader*. Toman la información y la presentan al usuario (HTML, etc.).

El sistema de configuración de URLs también forma parte de esta capa.

Control

En esta capa encontramos la lógica de la aplicación, el programa. La capa de presentación depende de esta capa.

En Django las *views* y *manipulators* representan esta capa.

Mediator

Maneja la interacción entre el subsistema *entity* y *foundation*. En él se realiza el mapeo objeto-relacional a cargo del motor de Django.

Entity

Maneja los objetos de negocio. El mapeo objeto-relacional de Django permite escribir objetos de tipo *entity* de una forma fácil y estándar.

Foundation

Su principal tarea es la de manejar a baja nivel el trabajo con la base de datos. A este nivel provee soporte para varias bases de datos y otras están en etapa de pruebas.

2.2.4. Soporte de bases de datos

La base de datos recomendada es *PostgreSQL*[6], pero *MySQL*[7] y *SQLite3*[8] también son soportadas por Django.

[6] http://www.postgresql.org.es/
[7] https://www.mysql.com/
[8] https://www.sqlite.org/

Una vez que se han creado los modelos de datos, Django proporciona una abstracción de la base de datos a través de su API. Permite crear, recuperar, actualizar y borra objetos.

El usuario también podrá ejecutar sus consultas SQL directamente.

En el modelo de datos de Django, una clase representa un registro de una tabla en la base de datos y cada instancia representa una de las filas de la tabla.

2.2.5. Soporte de servidores Web

Django incluye un servidor propio para realizar pruebas y trabajar en la etapa de desarrollo.

Para la etapa de producción se recomienda servidores como *Apache 2* con *mod_python*. Aunque Django puede correr sobre gran multitud de servidores como *FastCGI*, *SCGI* en Apache o *Lighttpd*, gracias a que soporta la especificación WSGI.

2.2.6. Instalación

Django [6] [7] está disponible bajo la licencia libre BSD. Django requiere Python versión 2.4 o superior pero no depende de otras librerías de Python.

Hay que configurar una base de datos. En caso de que la versión de Python instalada sea 2.5 o superior no será necesario. En caso contrario, o si se quiere trabajar con un gestor de bases de datos como *PostgreSQL*, *MySQL* u *Oracle* es necesario instalar la base de datos.

Para instalar Django es necesario borrar cualquier versión anterior antes de hacer la nueva instalación.

Existen tres formas sencillas de instalar Django:

- Instalar una versión de Django provista por la distribución de tu sistema operativo. Esta opción es la más rápida para lo que tienen sistemas operativos que distribuyen Django.

- Instalar una versión oficial publicada. Este es el mejor método para usuarios que quieren una versión estable y no les importa utilizar una versión de Django ligeramente más antigua.
  La última versión oficial de Django es la 1.8.5. Hay que descargar el paquete *Django-1.8.5.tar.gz* de la página:
  www.djangoproject.com/download/1.8.5/tarball/

- Instalar la última versión de desarrollo. Esta opción es la mejor para usuarios que quieran las ultimas características y que no tienen miedo a utilizar código nuevo.

2.2.7. Formato de una aplicación Django

Para crear el proyecto en Django hay que abrir una consola de comandos, situarse en la carpeta donde se quiere crear el proyecto y escribir:

```
> django-admin.py startproject (nombre del proyecto)
```

Ya estaría el proyecto creado.

Cada proyecto necesita aplicaciones donde gestionar los modelos y las vistas. Un proyecto puede tener muchas aplicaciones.

Para crear la primera aplicación del proyecto, desde la consola de comandos y en el directorio del proyecto hay que escribir:

```
> python manage.py startapp (nombre de la aplicación)
```

La estructura de ficheros quedaría de la forma mostrada en la siguiente figura:

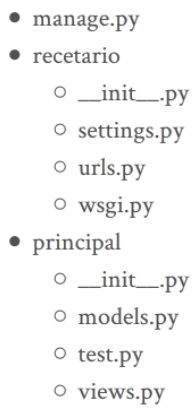


Figura 2. 3: Estructura de ficheros en Django

Donde ”recetario” se corresponde con el nombre del proyecto y “principal” se corresponde con el nombre de la aplicación.

2.3. Matplotlib

2.3.1. Introducción

Matplotlib [8] es una librería de código abierto creada para Python por John Hunter, la cual posee un conjunto de herramientas para poder hacer gráficas en 2D y 3D.

Permite interaccionar con librerías como qt, Gtk, entre otras para poder hacer interfaces de usuario personalizadas.

2.3.2. Características

Principales características de la librería Matplotlib [9]:

- Permite visualizar datos de forma muy rápida.
- Permite obtener gráficas de calidad para publicaciones.
- Lo podemos configurar con nuestras preferencias:
    - En Linux se puede configurar el usuario y el sistema.
    - En Windows se puede configurar el sistema.
    - Se puede configurar la sesión.

      ```
      >> rcParams
      ```

    - Se puede configurar el script.
- El principal submódulo para dibujar *pyplot*.

2.3.3. Instalación

Para instalar Matplotlib [10] es necesario ir a la página http://matplotlib.org/ y seleccionar la sección de descargas.

Una vez en la sección de descargas se debe escoger la versión que se desea instalar en el PC.

Se descarga el archivo en el directorio deseado y se procede a su instalación.

Una vez finalizada la instalación será necesario instalar algunos complementos a Python para que funcione correctamente.

- Importar la librería Matplotlib de la siguiente manera:

  ```
  import matplotlib.pyplot as plt
  ```

- Desde la consola de comandos ejecutar las siguientes instrucciones:

  ```
  > pip install six
  > pip install python-dateutil
  > pip install pyparsing
  ```

Quedaría la instalación completada.

2.3.4. Pyplot

Pyplot es un submodulo de *Matplotlib* que nos permite dibujar de forma sencilla. *Pyplot* permite elegir entre diversos elementos para pintar el gráfico que se desee.

A continuación se enumeran propiedades del gráfico que se pueden elegir con *Pyplot*:

Colores

Se pueden escribir letras que indican colores, nombres de colores, código hexadecimal se puede usar la *keyword* color.

```
plt.plot (x, y, color = 'blue')

plt.plot (x, y, 'b')

plt.plot (x, y, 'blue')

plt.plot (x, y, '#FF00FF')

plt.plot (x, y, color = '#FF00FF')
```

Color abbreviation	Color Name
b	blue
c	cyan
g	green
k	black
m	magenta
r	red
w	white
y	yellow

Figura 2. 4: Definición de colores en Pyplot [9]

Marcadores

Las líneas y marcadores se pueden controlar mejor con *keywords*.

```
plt.plot (x, color = 'b', linestyle = 'dashdot', linewidth = 4,
marker = 'o', markerfacecolor = 'red', markeredgecolor =
'black', markeredgewidth = 4, markersize =12)
```

Keyword argument	Description
color or c	Sets the color of the line; accepts any Matplotlib color format.
linestyle	Sets the line style; accepts the line styles seen previously.
linewidth	Sets the line width; accepts a float value in points.
marker	Sets the line marker style.
markeredgecolor	Sets the marker edge color; accepts any Matplotlib color format.
markeredgewidth	Sets the marker edge width; accepts float value in points.
markerfacecolor	Sets the marker face color; accepts any Matplotlib color format.
markersize	Sets the marker size in points; accepts float values.

Figura 2. 5: Keywords en Pyplot [9]

Al igual que con los colores, hay muchas formas de definir los marcadores.

Style abbreviation	Style
-	solid line
--	dashed line
-.	dash-dot line
:	dotted line

Figura 2. 6: Estilos en Pyplot [9]

Marker abbreviation	Marker style
.	Point marker
,	Pixel marker
o	Circle marker
v	Triangle down marker
^	Triangle up marker
<	Triangle left marker
>	Triangle right marker
1	Tripod down marker
2	Tripod up marker
3	Tripod left marker
4	Tripod right marker
s	Square marker
p	Pentagon marker
*	Star marker
h	Hexagon marker
H	Rotated hexagon marker
+	Plus marker
x	Cross (x) marker
D	Diamond marker
d	Thin diamond marker
|	Vertical line (vline symbol) marker
_	Horizontal line (hline symbol) marker

Figura 2. 7: Marcadores en Pyplot [9]

Etiquetas para los ejes

Las etiquetas se controlan mediante *xticks* e *yticks*.

```
plt.plot (range(5))

plt.xticks (range(5), ('x1', 'x2', 'x3', 'x4', 'x5'))
```

Tipos de gráficos

- Histogramas

```
y = np.random.randn (1000)
plt.hist (y, 25)
```

- Circulares

```
aficion = [3000, 3000, 10]
etiquetas = ['Madrid', 'Barcelona', 'Getafe']
plt.pie (afición, labels = etiquetas)
```

- Scatter

```
x = arrange (1000)
y = np.random.randn (1000)
plt.scatter (x, y)
```

A continuación se representan todos los tipos de gráficos que se pueden pintar con *Matplotlib*.

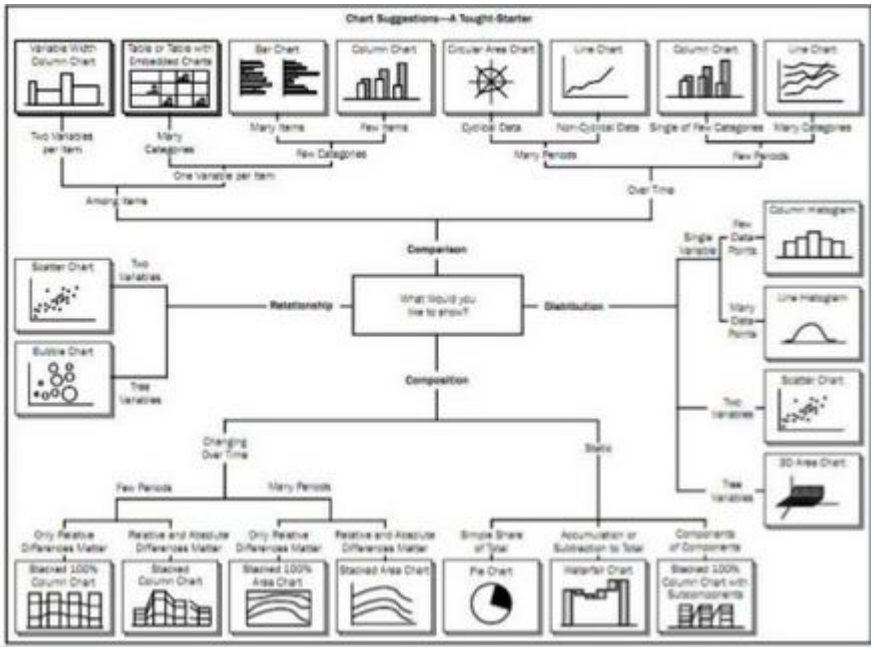


Figura 2. 7: Tipos de gráficos en Pyplot [9]

Subplots

Consiste en varios gráficos en el mismo 'tapiz'.

Subplot (número de filas, número de columnas, orden).

```
plt.subplot (211)

plt.plot (arange (10))

plt.subplot (212)

plt.scatter (arrange (0, 10, -1))
```

2.4. API REST de Twitter

En la API REST de Twitter [11] podemos consultar toda la información a la que un usuario puede acceder a través de la interfaz web de twitter.com. Para acceder a muchos de estos recursos se necesita autenticación. Además el formato en el que se obtienen los datos puede ser XML, ATOM, JSON o RSS. En nuestro caso queremos acceder a los tweets a través de la API REST y esto requiere autenticación por OAuth. Este protocolo se detalla en siguientes apartados de esta misma sección de la memoria.

Es importante indicar que la API REST de Twitter tiene una limitación en cuanto al número de peticiones que un usuario puede hacer. Si el usuario está autenticado el límite es de 350 peticiones/hora, en caso de que el usuario no esté autenticado el límite es más restrictivo y es de 150 peticiones/hora. El contador de peticiones se resetea cada hora.

2.4.1. Recursos disponibles a través de la API REST de Twitter

La API REST de Twitter dispone de numerosos recursos pero en este documento sólo se van a especificar los son útiles para este proyecto.

OAuth

El protocolo OAuth se utiliza para la autenticación de usuarios.

- `POST oauth/request_token` - Permite a una aplicación solicitar un token temporal para el protocolo OAuth.
- `POST oauth/accesst_token` - Obtiene su token de acceso.
- `GET oauth/authenticate` - Permite a una aplicación consumidora usar token temporal para solicitar la autenticación.
- `GET oauth/authorization` - Permite a una aplicación consumidora usar token temporal para solicitar la autorización.

Search API

La API Search de Twitter es parte de la API REST. Esta API permite la consulta de índices de Tweets recientes o populares.

- `GET search/tweets` – Devuelve una colección de Tweets aplicando los criterios de búsqueda especificados en una query. A esta query de búsqueda se le pueden pasar varios parámetros como palabras clave que se desea que contengan los tweets seleccionados o el número de tweets a recuperar.

2.5. Protocolo OAuth

OAuth (*Open Authorization*) [12] [13] [14] es un protocolo que permite flujos simples de autorización para sitios web o aplicaciones informáticas. Se trata de un protocolo que permite autorización segura de una API de modo estándar y simple para aplicaciones de escritorio, móviles y web.

Este protocolo permite a un usuario del sitio A compartir su información en el sitio A (proveedor de servicio) con el sitio B (consumidor) sin compartir toda su identidad. Para desarrolladores de proveedores de servicio, OAuth proporciona a los usuarios un acceso a sus datos al mismo tiempo que protege las credenciales de su cuenta.

En el proceso OAuth intervienen tres actores, que son los siguientes:

1. Proveedor de servicios, que es, el servicio de autenticación externo (en este caso Twitter).
2. Consumidor, es el servicio al que el usuario quiere acceder usando una cuenta externa (la del proveedor de servicios).
3. Usuario

La API de Twitter posee métodos para poder utilizar el protocolo OAuth en aplicaciones desarrolladas por terceros.

Se debe distinguir entre la autorización y la autenticación. Los dos procesos pueden realizarse a partir de la API REST de Twitter.

La autorización se centra en un escenario en el que el consumidor quiere acceder a información del usuario en el proveedor de servicios y el usuario autoriza dicho acceso. En cambio, la autenticación, permite al usuario acceder a información y recursos en el consumidor, pero autenticarse con su cuenta del proveedor de servicios.

En el caso de la autenticación se otorga permiso para la realización de una acción, mientras que en la autorización se valida la identidad de un usuario.

Con el protocolo OAuth los pasos que hay que dar para la autenticación y autorización son los mismos. En cambio en el caso de la API de Twitter cambiarían las llamadas a las URLs. Para la autorización esta el método `GET oauth/authorize` y para la autenticación el método `GET oauth/authenticate`.

2.5.1. Flujo de trabajo de OAuth en Twitter

Antes del proceso de autenticación por OAuth, el usuario debe haber dado de alta la aplicación consumidora en Twitter. Así se dispondrá de los parámetros necesarios en el flujo de trabajo con OAuth, el *consumer_key* y el *consumer_secret*.

La autenticación por OAuth se realiza en tres pasos:

1. El consumidor obtiene un token temporal.
2. El usuario autoriza el token.
3. Se intercambia el token temporal por un token de acceso permanente.

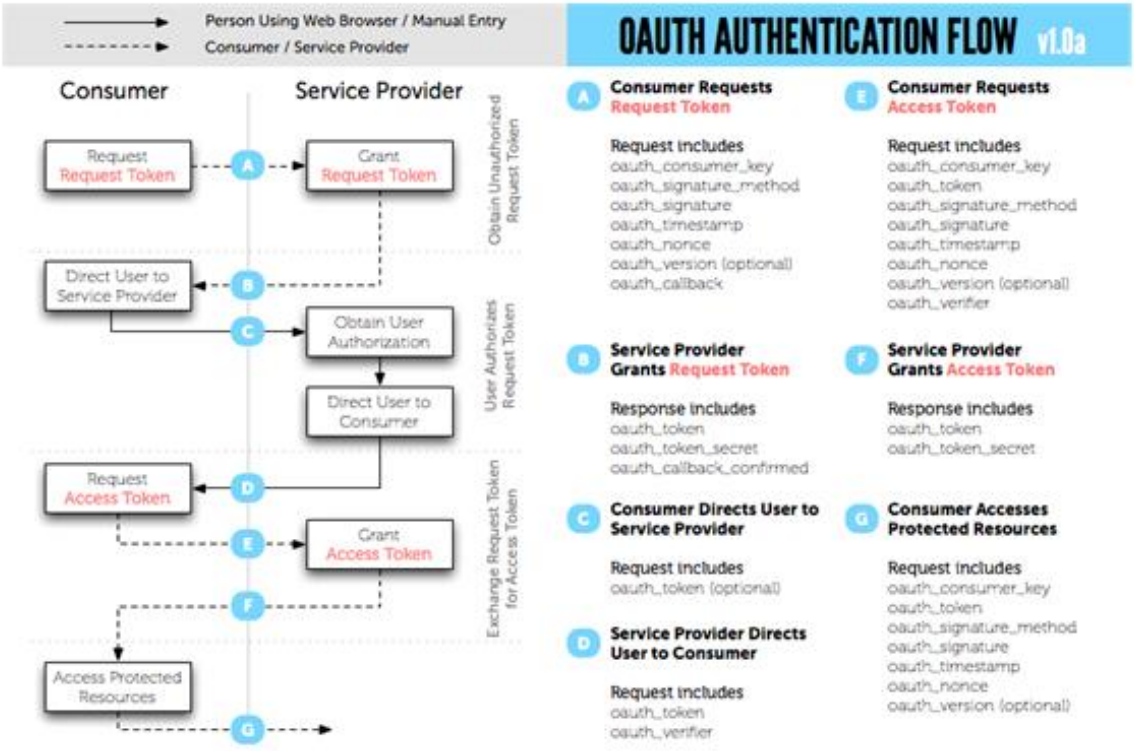


Figura 2. 8: Flujo de trabajo para OAuth [12]

2.6. MeaningCloud

2.6.1. Introducción

MeaningCloud [15] [16] es la manera más sencilla, potente y asequible de extraer el significado de todo tipo de contenido no estructurado: conversaciones sociales, artículos, expedientes, etc.

Ofrece APIs Semánticas en modo SaaS[9] de Deadelus. Esta dirigido a desarrolladores e integradores. Cuenta una serie de APIs estándar de alto nivel específicas de la aplicación para diversos sectores y escenarios de uso, liberando así a los usuarios de la complejidad técnica de estos desarrollos y reduciendo su *time-to-market*.

2.6.2. Características

MeaningCloud posee una serie de características que hace que sea una de las maneras más sencillas de extraer el significado de un determinado contenido.

- **Potente**
  Se debe a que combina las tecnologías más avanzadas para proporcionar las funcionalidades más avanzadas, como por ejemplo:
  - Análisis de sentimiento orientado a aspectos.
  - Procesamiento del lenguaje de los medios sociales.

- **Personalizable**
  Proporciona interfaces gráficas para permitir al usuario personalizar fácilmente el sistema usando sus propios diccionarios y modelos.

- **Fácil de usar e integrar**
  Es posible utilizarlo desde add-in de Excel, se puede integrar sin programas mediante plug-ins, se puede desarrollar sobre SDK y servicios web estándar. Es una plataforma abierta, fácil de aprender y de usar.

- **Sin compromisos**
  No necesita software que instalar, ni tampoco infraestructuras que desplegar. Proporciona toda la fiabilidad y escalabilidad en la nube y la posibilidad de probarlo de forma gratuita.

- **Múltiples idiomas**
  Analiza los contenidos en idiomas como: Español, Inglés, Francés, Portugués o Italiano.

- **Asequible**
  Sólo hay que pagar por lo que usas. No hay cuotas de activación, ni de permanencia.

2.6.3. Uso

Detalle de donde y para que se puede usar MeaningCloud:

- **Análisis de la Voz del Cliente (VoC) y Gestión de la Experiencia de Cliente**
  Analiza el feedback de los clientes en formato libre a través de cualquier canal (email, call center, encuestas, medios sociales) y gestiona su experiencia en todos sus puntos de contacto con la empresa.

[9] Software as a Service

- **Publicación y monetización de contenidos**
  Gestiona un archivo histórico, produce nuevos contenidos de mayor calidad y monetizados mejor mediante la personalización y la publicidad enfocada.

- **Análisis de medios sociales**
  Entiende conversaciones en foros y redes sociales en varios idiomas, con volúmenes masivos y en tiempo real. Descubre el sentimiento del mercado, escucha la voz del ciudadano y detecta emergencias y amenazas para la seguridad.

- **Codificación y gestión de expedientes**
  Codifica y clasifica automáticamente expedientes de todo tipo (médicos, judiciales, administrativos, etc.) para gestionarlos más eficientemente, elaborar informes y detectar tendencias y relaciones.

2.6.4. APIs

MeaningCloud cuenta con una serie de APIs cada una de las cuales realiza un análisis diferente del texto.

A continuación se enumeran y describen las diferentes APIs:

Extracción de Topics

Extrae información relevante de textos, como entidades con nombre (personas, lugares, organizaciones, etc.) y conceptos, así como otros datos de importancia (fechas, expresiones de tiempo, cantidades monetarias, etc.)

Clasificación de Texto

Categoriza textos automáticamente en una clasificación jerárquica o taxonomía..

Análisis de Sentimiento

Analiza el sentimiento y etiqueta textos con la indicación de su polaridad, subjetividad, ironía y expresión de desacuerdo.

Identificación de idioma

Detecta automáticamente el idioma de un texto obtenido de cualquier tipo de fuente.

Lematización, Análisis Morfológico y Sintáctico

Realiza el análisis lingüístico detallado para varios idiomas incluyendo la extracción del lema y el análisis morfo-sintáctico de cada palabra (*Part of Speech tagging*) además del análisis sintáctico basado en constituyentes para las oraciones.

Corrección de textos

Detecta automáticamente errores ortográficos, gramaticales y de estilo en español.

Reputación Corporativa

Esta API proporciona etiquetado semántico y de sentimiento de las dimensiones apropiadas para el análisis de la reputación corporativa.

Clustering de texto

Solución para realizar automáticamente *clustering* de documentos de cara a agruparlos por similitud y descubrir temas significativos.

2.6.5. Integraciones

MeaningCloud tiene dos integraciones en su versión gratuita. Se detallan a continuación:

MeaningCloud para Excel

Permite realizar de manera sencilla análisis de texto desde una hoja de cálculo.

Plugin para GATE

Permite acceder a Textalytics desde GATE (General Architecture for Text Engineering), una plataforma de Ingeniería Lingüística ampliamente usada.

2.7. CSS

El lenguaje CSS [17] [18] (Cascading Style Sheets) se utiliza para definir la parte de presentación de un documento estructurado escrito en HTML o XML.

El organismo que se encarga de elaborar las especificaciones y los estándares de las hojas de estilo es el W3C (World Wide Web Consortium).
Este lenguaje surgió con el propósito de separar la estructura de un documento de su presentación.

2.7.1. Sintaxis

La sintaxis del lenguaje CSS es muy sencilla. Hace uso de palabras clave que especifican los nombres de los selectores, propiedades y atributos de los que quiere modificar la presentación.

Dentro de estas hojas de estilo se definen los bloques encerrados entre llaves donde se definen las propiedades, con el formato `propiedad:valor` de cada uno de los selectores.

Los selectores a los que se hace referencia en cada bloque del CSS, indicarán los elementos que se verán afectados en función de su tipo, nombre (name), ID, clase (class), posición dentro del DOM (Document Object Model), etc.

2.7.2. Uso

CSS se puede usar de tres formas distintas a la hora de dar formato a un documento:

1. Hoja de estilo CSS externa. El código de la hoja de estilo está en un fichero con extensión `.css`. Este modo es el más potente ya que separa totalmente las reglas de estilo del documento estructurado que es la filosofía con la que surge CSS.

   La hoja de estilo se asocia al documento mediante la etiqueta `<link>`, esta etiqueta irá dentro del elemento `<head>` del documento HTML.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN">
<html>
  <head>
    <title>Título</title>
    <link            rel="stylesheet"            type="text/css"
href="http://www.w3.org/css/officeFloats.css" />
  </head>
  <body>
  .
  .
  .
  .
  </body>
</html>
```

2. Incluyendo la hoja de estilo dentro del documento al que se le quiere dar formato. La hoja de estilos se debe definir dentro del elemento `<head>` dentro del documento HTML y definida por la etiqueta `<style>`.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN">
<html>
  <head>
    <title>hoja de estilo interna</title>
    <style type="text/css">

      body {
        padding-left: 11em;
        font-family: Georgia, "Times New Roman", serif;
        color: red;
        background-color: #d8da3d;
      }

      h1 {
        font-family: Helvetica, Geneva, Arial, sans-serif;
      }
```

```
    </style>
  </head>
  <body>
    <h1>Aquí se aplicará el estilo de letra para el Título</h1>
  </body>
</html>
```

3. Aplicando estilos directamente sobre los elementos del documento estructurado que lo permita. Se hace mediante el atributo `<style>` dentro de la etiqueta `<body>`. Este tipo de definición de estilo mezcla el contenido del documento estructurado con la presentación, esto hace que se pierdan las ventajas que ofrece el CSS.

2.7.3. Ejemplos y normas básicas

En el ejemplo siguiente se indica que el selector `<h1>` de un documento HTML se verá afectado por la regla de estilo que va entre las llaves:

```
h1{color: red;}
```

Si queremos agrupar varios selectores para que se les apliquen las mismas reglas de estilo los separaremos por comas:

```
h1, h2, h3 {
  color: red;
}
```

Que es lo mismo que:

```
h1 {color: red;}
h2 {color: red;}
h3 {color: red;}
```

Dentro de un selector podrán agruparse varias propiedades que se quieran cambiar, pero deberán separarse por un punto y coma:

```
p {text-align:center;color:red}
```

Aunque normalmente se describe una propiedad por línea:

```
h1 {
  padding-left: 11em;
  font-family: Georgia, "Times New Roman",Times, serif;
  color: red;
  background-color: #d8da3d;
}
```

En el caso de que el valor de una propiedad de estilo esté formado por más de una palabra deberá ir entre comillas:

```
p {font-family: "sans serif";}
```

2.8. Estudios de mercado

El objetivo de este proyecto es obtener información de las redes sociales para elaborar un estudio de mercado.

2.8.1. Definición y análisis

Un estudio de mercado [19] [20] consiste en una iniciativa empresarial para conocer la viabilidad comercial de un proyecto.

El principal objetivo de un estudio de mercado es recoger información y hacer un análisis de esa información para conocer las condiciones del mercado, tomar decisiones y anticiparse a la evolución del mercado.

Para el estudio de mercado se elaboran tres análisis.

- **Análisis de consumidores**
  La finalidad de este análisis es conocer el comportamiento de los consumidores y sus preferencias a la hora de consumir un producto. Esto es para detectar las necesidades del consumidor y poder satisfacerlas. Las encuestas, paneles, entrevistas o las reuniones en grupo son algunas de las formas de obtener datos.

- **Análisis de competencia**
  En este análisis se estudia con que recursos se cuenta y cuál es el producto que se ofrece. Una vez que se conocen las fortalezas y debilidades se debe hacer un análisis externo del mercado para ver en qué aspectos se puede mejorar con respecto a la competencia.

- **Estrategia**
  La estrategia es un plan que marca el rumbo a seguir por la empresa. Este plan se elabora en base a los objetivos, recursos, estudios de mercado y competencia.
  Existen dos estrategias posibles: liderazgo en costo, que consiste en mantener el liderazgo aventajando a la competencia en materia de costos, y diferenciación, consiste en elaborar un producto que aventaje a sus competidores en diferentes características (diseño, funcionalidad, imagen,…).

2.8.2. Herramientas online para el estudio de mercado

Este proyecto consiste en una aplicación web para elaborar estudios de mercado pero ya existen numerosas herramientas online que generan estudios de mercado.

A continuación vemos algunos ejemplos.

- **Google Trends**
  Es una herramienta que proporciona Google y consiste en el estudio de keywords o palabras clave para descubrir tendencias de consumo.
  Es un servicio gratuito y fácil de utilizar.

- **Keyword Tool External**
  El objetivo de esta herramienta es calcular el volumen de tráfico teniendo en cuenta las palabras más usadas en el buscador de Google.

- **Feebbo**
  Esta herramienta genera informes a través de encuestas online entre sus usuarios. El coste de esta herramienta es de unos 3 euros por persona consultada.

- **TrendWatching**
  Publica mensualmente estudios de tendencias de los consumidores de forma gratuita. Sus datos son a gran escala pero permiten obtener una visión global de las tendencias del mercado. Existe una versión Premium que presenta información más detallada.

- **TusEncuentas**
  Es una herramienta para realizar encuestas online. Permite realizar diferentes tipos de encuestas con las preguntas elaboradas por el cliente. Es un servicio gratuito que no tiene límite de preguntas ni de usuarios.

- **Encuesta fácil**
  Es una herramienta para realizar encuestas online. Permite diseñar cuestionarios a medida, recopilar resultados y analizarlos en tiempo real. Es un servicio gratuito para las 100 primeras respuestas.

2.8.3. Escucha social

Todas las técnicas mencionadas en el apartado anterior han ido evolucionando a la vez que las redes sociales. Gracias a esto es posible conocer las tendencias de consumo a partir de las pautas de conducta que muestran los usuarios en las redes sociales como Facebook, Twitter, Tuenti, Badoo…

La escucha social consiste en la lectura de comentarios, opiniones o menciones a una determinada marca o producto en las redes sociales.

En Facebook ya es posible hacer un seguimiento de los usuarios a través de las Fan Page.

Para la escucha social ya existen algunas herramientas:

- **AGNA**
  Es la herramienta pionera en este campo. Surgió en 2008 a la par que muchas redes sociales.

Esta herramienta se basa en el análisis secuencial y la sociometría. Los análisis que ofrece esta herramienta están basados en métodos matemáticos usados en psicología social, sociología, antropología y etología.

- **Socilyzer**
  Permite hacer análisis básicos a través de cuestionarios.

- **Pollowers**
  Plataforma para el análisis en Twitter. Esta herramienta permite realizar encuestas que se publicaran en el timeline de Twitter y que los usuarios respondan con un *reply*.

- **Twylah**
  Plataforma para el análisis de Twitter. Esta herramienta utiliza los temas más populares de Twitter y crea memes[10] a partir de ellos. Se trata de una herramienta de curación de contenidos pero que resulta muy útil a la hora de identificar tendencias.

[10] Es una idea, concepto, situación, expresión o pensamiento manifestado en cualquier tipo de medio virtual

Capítulo 3

Requisitos de diseño

En este capítulo de la memoria vamos a hablar de los requisitos definidos para el funcionamiento de la aplicación.

Se empieza describiendo cual es el funcionamiento definido para esta aplicación y a continuación se comentan las decisiones de diseño que se han tomado para que la aplicación funcione de la manera deseada.

3.1. Funcionamiento de la aplicación

El funcionamiento de la aplicación consiste en hacer una comparativa de dos marcas o productos en base a las opiniones recogidas en una red social, en este caso Twitter.

La aplicación debe recoger el nombre de dos marcas o productos y con estas marcas hacer una consulta a Twitter obteniendo un determinado número de tweets que hagan referencia a esas marcas o productos.

Una vez se han obtenidos los tweets se procede a realizar un análisis del texto de cada tweet para conocer el sentimiento del tweet, extraer conceptos, clasificar en temas, etc. Este análisis del texto se realizara mediante la herramienta MeaningCloud que proporciona una serie de APIs para obtener y clasificar la información obtenida de un texto.

Cuando se finalice con el análisis del texto y se haya clasificado se procederá a mostrar los resultados obtenidos del análisis.

El diagrama de la aplicación será el siguiente.

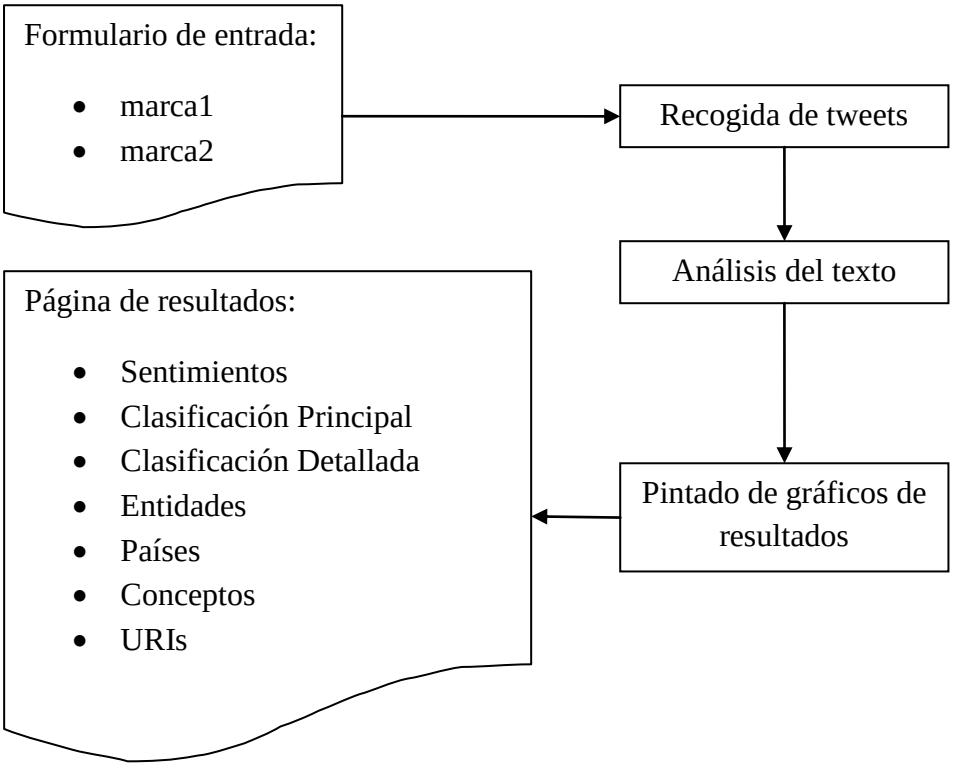


Figura 3. 1: Diagrama de navegación

3.2. Decisiones de diseño

Tras la descripción del funcionamiento se ha realizado un análisis técnico de la aplicación y se han establecido los siguientes criterios.

1. Será una aplicación web, por tanto, correrá en un servidor.

2. Debe darse de alta la aplicación en Twitter a través de la página https://apps.twitter.com/. Es necesario disponer de:
   - consumer_key
   - consumer_secret

3. El usuario que desee usar la aplicación debe tener una cuenta de usuario en Twitter. Necesita tener :
   - access_token_key
   - access_token_secret

4. El usuario también deberá tener abierta una cuenta en MeaningCloud. Necesita tener:
   - license_key

5. El usuario podrá elegir por parámetro de entrada el número de tweets que desea consultar a la API de Twitter.

6. El usuario podrá elegir por parámetro de entrada el número de resultados que quiere mostrar en los gráficos de entidades, conceptos y URIs.

7. El usuario podrá introducir por parámetros el nombre de dos marcas o productos para realizar el análisis y comparación de cada una.

8. Si alguno de los parámetros no es introducido por el usuario, se mostrará un mensaje de error por pantalla y se volverá al inicio.

9. Si al autenticarse en Twitter se produjera un error, se mostrará un mensaje de error por pantalla y se volverá al inicio.

10. Si al autenticarse en MeaningCloud se produjera un error, se mostrará un mensaje de error por pantalla y se volverá al inicio.

11. La aplicación deberá recoger de Twitter el número de tweets que el usuario indique por parámetro.

12. La aplicación deberá analizar cada uno de los tweets recogidos y analizarlos con las APIs de MeaningCloud. Las APIs serán las de Topics Extraction, Text Classification y Sentiment Analysis.

13. Los resultados del análisis de los tweets mediante las APIs de MeaningCloud serán mostrados en una pantalla de salida en forma de gráficos.

Capítulo 4

Diseño de alto nivel

En este capítulo de la memoria hablaremos del diseño a alto nivel de la aplicación. Se hablará en detalle del diseño previo de la interfaz web de usuario y del resultado final en cuanto a la interfaz se refiere.

4.1. Diseño previo de la interfaz de usuario

La aplicación web desarrollada tiene una interfaz web muy sencilla. La aplicación tendrá una primera pantalla con un formulario para la recoger los parámetros de entrada necesarios para el funcionamiento de la aplicación y tendrá una página de resultados con distintos apartados.

Las pantallas de la aplicación serán implementadas en HTML y se utilizará un fichero CSS para dar formato a las páginas.

4.1.1. Acceso a la aplicación y pantalla de inicio

Si la aplicación web está corriendo en el servidor de pruebas de Django, se podrá acceder a ella mediante la URL: `http://127.0.0.1:8000/`en un navegador.

En caso de que la aplicación este corriendo en un servidor web local, se podrá acceder a ella mediante la URL: `http://localhost:8080/comparadorMarcas/` en un navegador. Y en caso de que la aplicación se encuentre en un servidor externo, en el navegador habría que introducir la URL: `http://{servidor}:{puerto}/comparadorMarcas/`

Como se ha mencionado anteriormente la pantalla de inicio tendrá un formulario para la recogida de parámetros de entrada.

El formulario estará dividido en las siguientes secciones:

- Datos Twitter
- Datos MeaningCloud
- Marcas
- Número de resultados

El diseño previo de la pantalla de inicio se muestra en la siguiente figura:

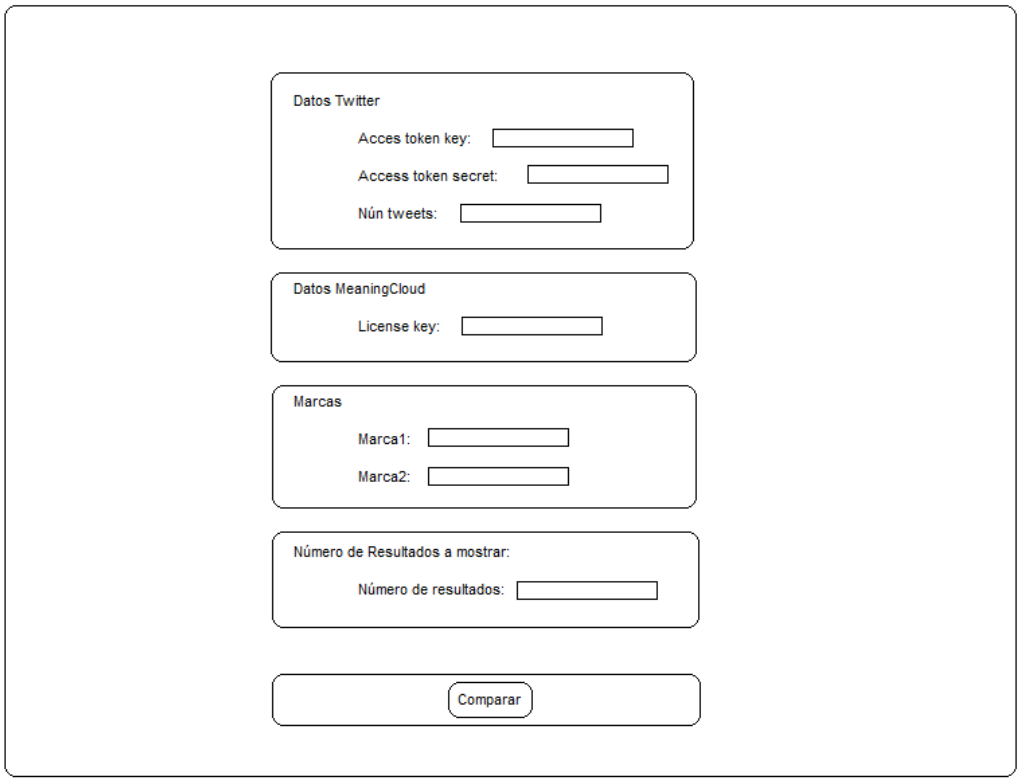


Figura 4. 1: Diseño de página de inicio

4.1.2. Pantalla de Resultados

Tras el análisis del texto, la aplicación genera una serie de gráficos que se mostrarán en una pantalla final dividida en una serie de secciones. Cada una de estas secciones tendrá dos bloques, cada uno de ellos para una marca.

La pantalla de resultados se divide en las siguientes secciones:

- Sentimientos
- Clasificación Principal
- Clasificación Detallada
- Entidades
- Países
- Conceptos
- URIs

El diseño previo de la pantalla de resultados de la aplicación se muestra en la siguiente figura:

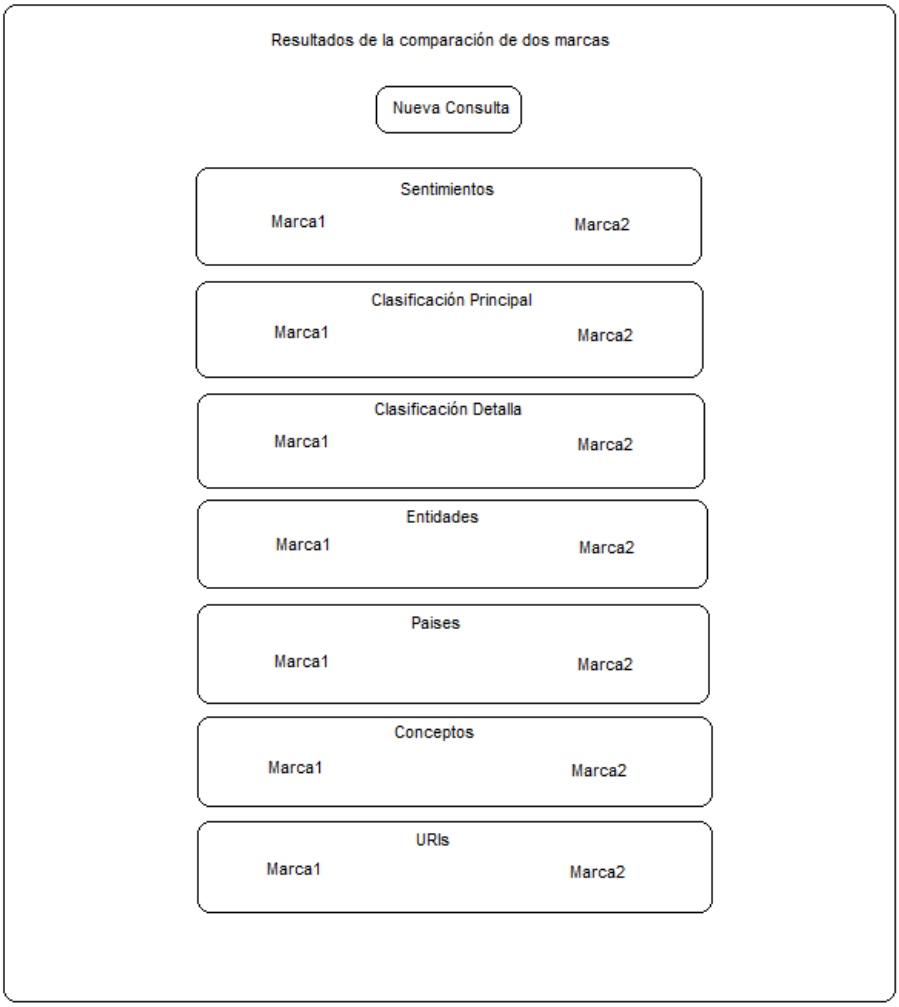


Figura 4. 2: Diseño de pantalla de resultados

4.2. Interfaz de usuario

Se va a proceder a mostrar el resultado final de la interfaz de usuario. A este resultado se ha llegado siguiendo el diseño previo y tras la implementación de la aplicación, que puede verse en detalle en el *Capítulo 5. Implementación del sistema*.

4.2.1. Pantalla de inicio

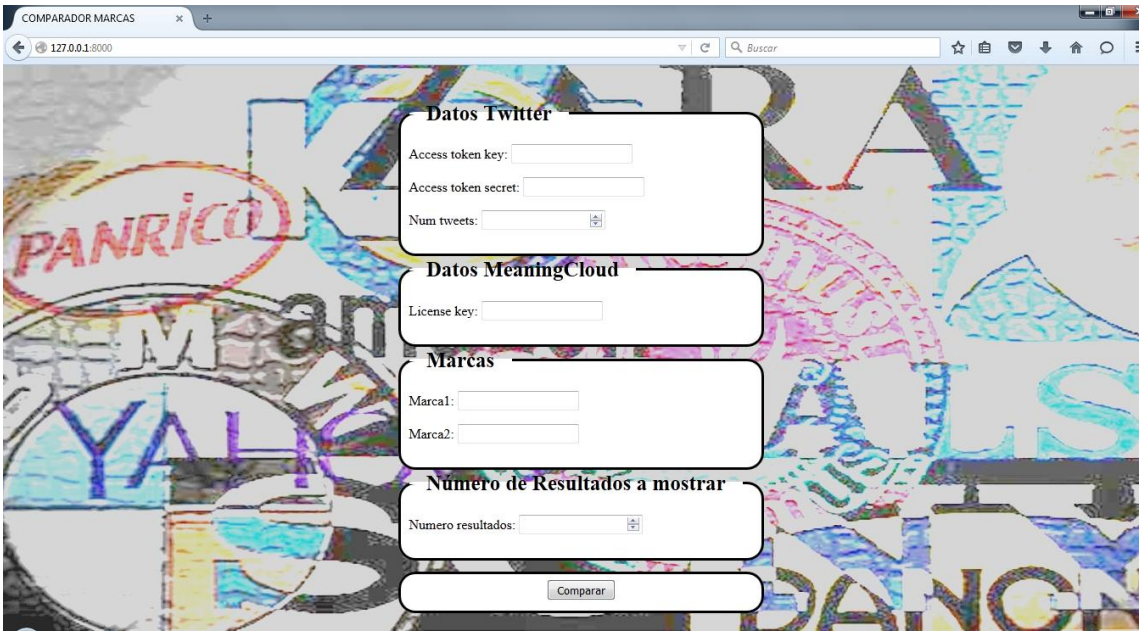


Figura 4. 3: Pantalla principal de la aplicación

Esta es la pantalla principal de la aplicación y en ella se deberá rellenar todos los campos del formulario:

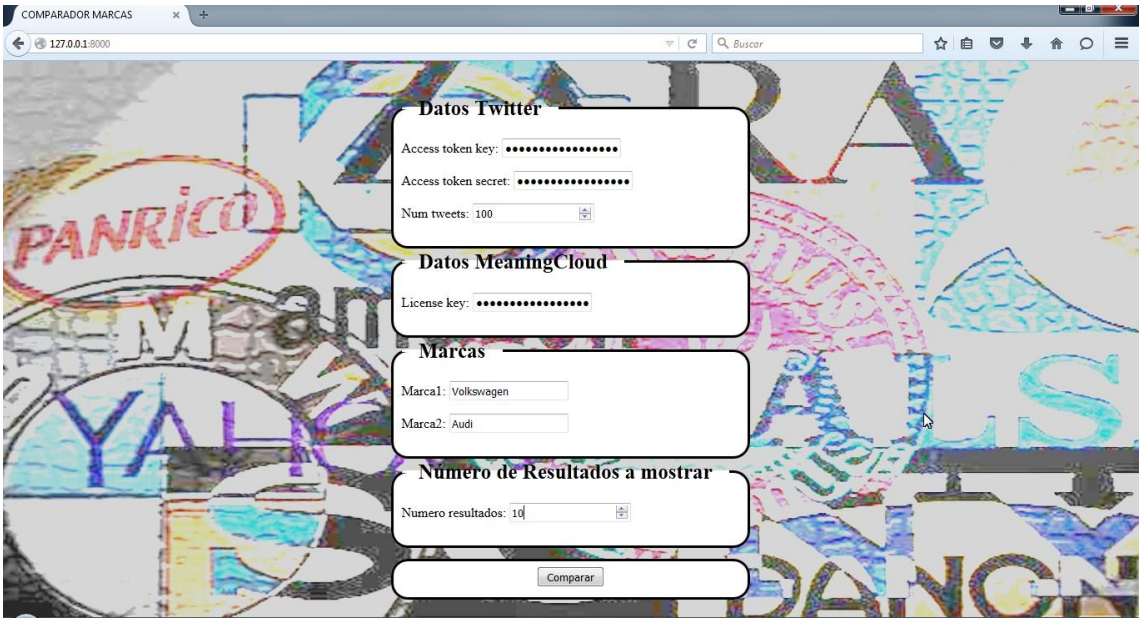


Figura 4. 4: Pantalla principal de la aplicación con formulario relleno

Una vez que los datos del formulario han sido rellenados se pulsa el botón `Comparar` para iniciar la consulta y el procesado del texto de los tweets. Esto nos llevará a la pantalla de resultados.

4.2.2. Pantalla de inicio en caso de error

Si al pulsar el botón `Comparar` hay algún parámetro incorrecto o se produce algún error en tiempo de ejecución se vuelve a la pantalla de inicio y se muestra un mensaje con el código de error y su descripción.

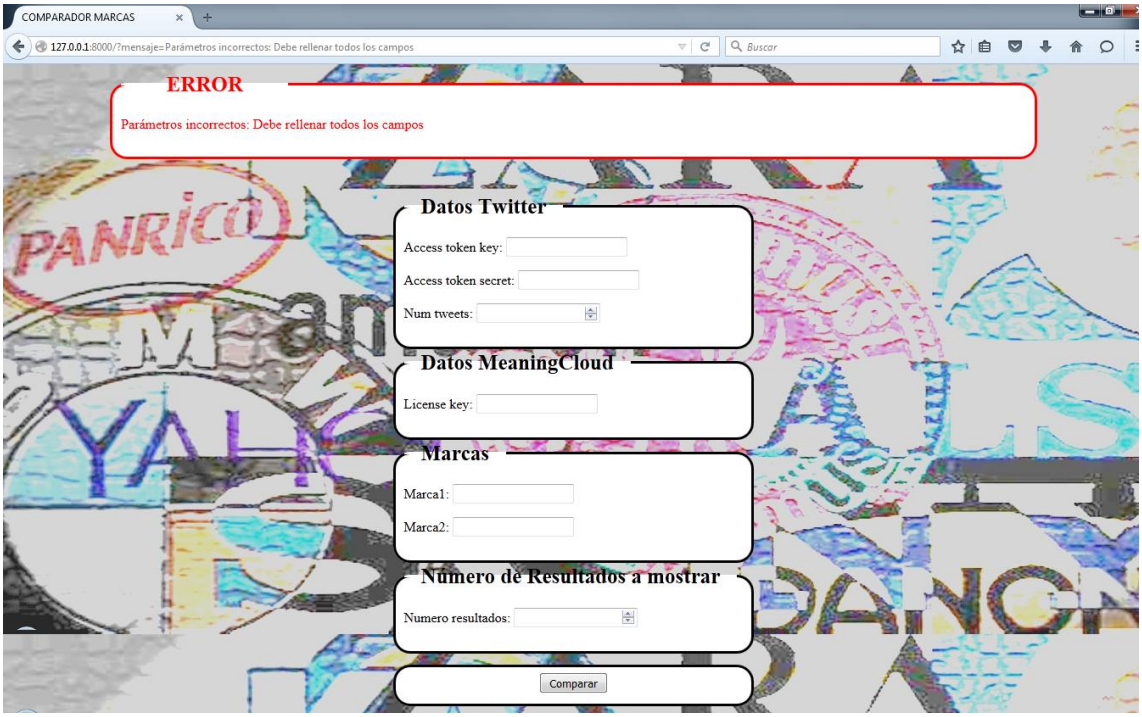


Figura 4. 5: Pantalla de inicio en caso de error por parámetros incorrectos

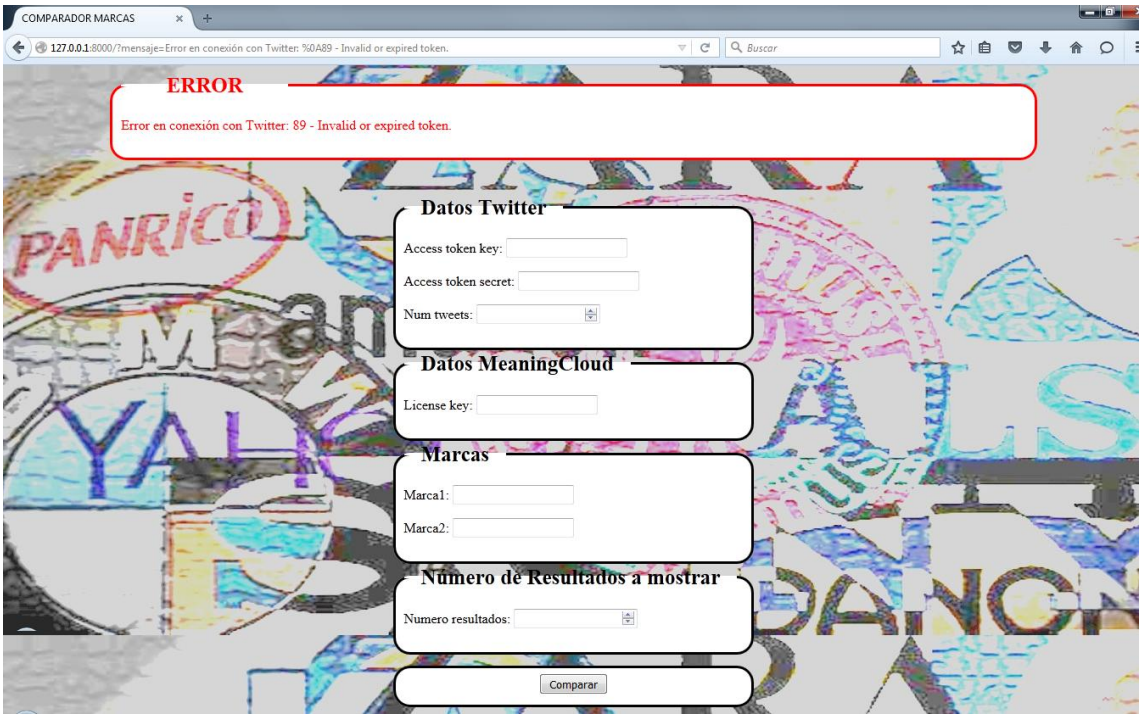


Figura 4. 6: Pantalla de inicio en caso de error en tiempo de ejecución

4.2.3. Pantalla de resultados

Una vez que se han hecho las consultas y se han analizado los tweets los resultados se muestran en una pantalla final que contiene los siguientes gráficos:

- Sentimientos
- Clasificación Principal
- Clasificación Detalla
- Entidades
- Países
- Conceptos
- URIs

La pantalla completa de resultados tiene el aspecto mostrado en la figura siguiente:

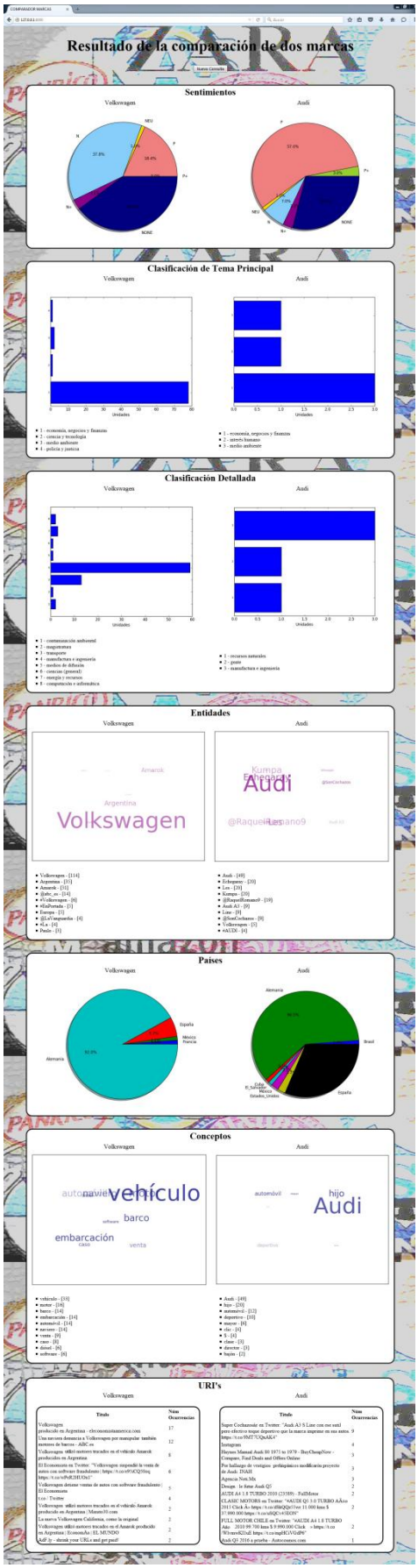


Figura 4. 7: Pantalla de Resultados completa

4.2.4. Pantalla de Resultados: Gráfico de Sentimientos

Este gráfico muestra los sentimientos de los usuarios de Twitter con respecto a determinadas marcas.

Ejemplo:

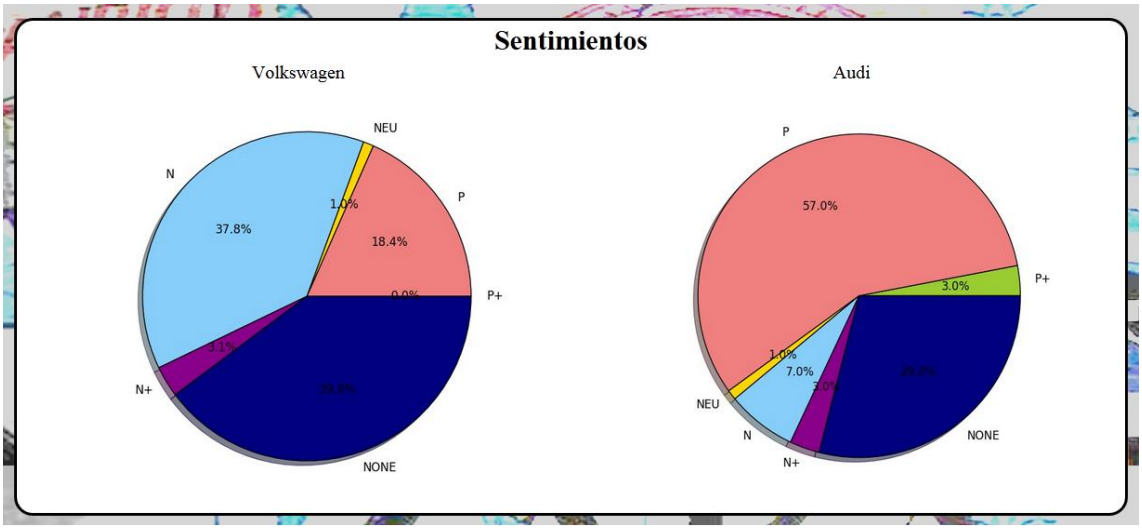


Figura 4. 8: Ejemplo de Gráfico de Sentimientos

Donde:

- P+: Muy positivo
- P: Positivo
- NEU: Neutral
- N: Negativo
- N+: Muy negativo
- NONE: No expresa sentimiento

Para el ejemplo se han consultado los últimos 100 tweets.

4.2.5. Pantalla de Resultados: Clasificación Principal

En este tipo de gráfico se muestra una clasificación principal de los tweets analizados por temas. Esta clasificación corresponde con el primer nivel de clasificación de IPTC.

Ejemplo:

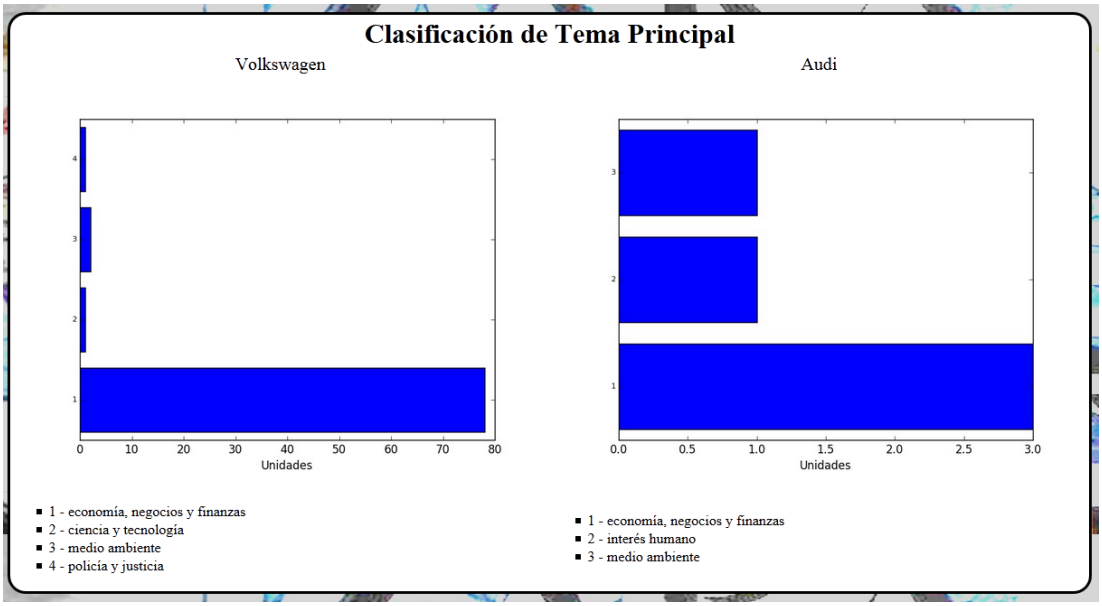


Figura 4. 9: Ejemplo de gráfico de Clasificación Principal

Para el ejemplo se han consultado los últimos 100 tweets.

4.2.6. Pantalla de Resultados: Clasificación Detallada

Al igual que el tipo de gráfico anterior este hace una clasificación por temas de los tweets analizados pero en este caso la clasificación es más detallada. Esta clasificación se corresponde con el segundo nivel de clasificación de IPTC.

Ejemplo:

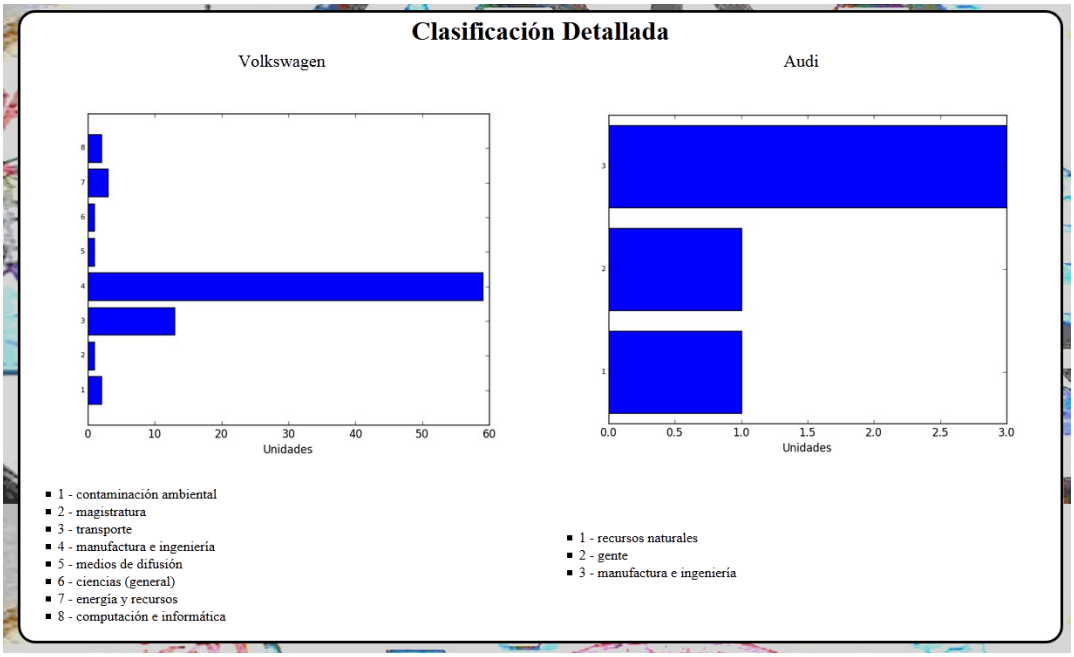


Figura 4. 10: Ejemplo de gráfico de Clasificación Detallada

4.2.7. Pantalla de Resultados: Gráfico de Entidades

En este gráfico se muestra las entidades que aparecen con más frecuencia en los tweets consultados.

Para este ejemplo se han consultado los últimos 100 tweets y se muestran las 10 entidades que aparecen con más frecuencia:

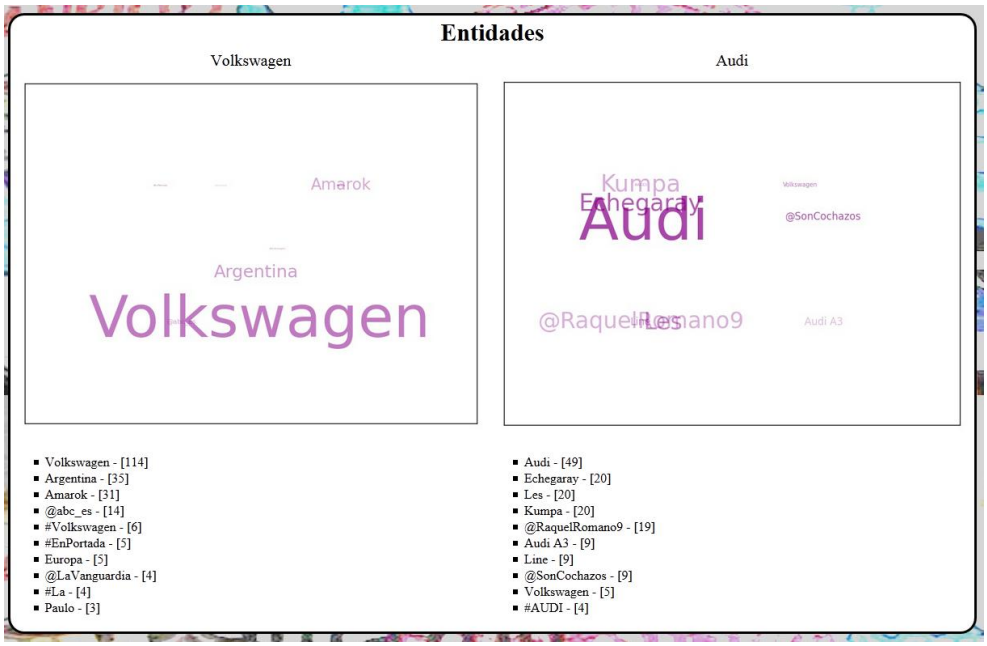


Figura 4. 11: Ejemplo de gráfico de Entidades

Cuantos menos resultados se muestren en el gráfico, más claro aparecerá este.

4.2.8. Pantalla de Resultados: Gráfico de Países

En este gráfico se muestran los países asociados a los tweets.

Ejemplo:

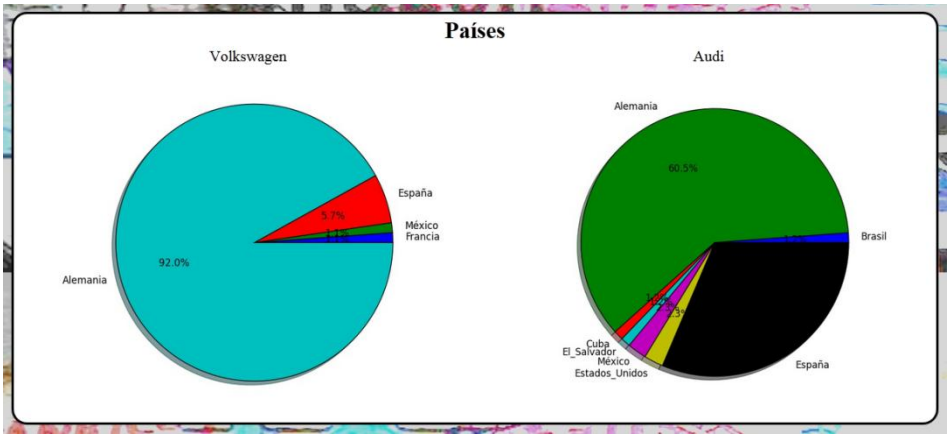


Figura 4. 12: Ejemplo de Gráfico de Países

Para el ejemplo se han consultado los últimos 100 tweets.

4.2.9. Pantalla de Resultados: Gráfico de Conceptos

En este gráfico se muestra los conceptos que aparecen con más frecuencia en los tweets consultados.

Para este ejemplo se han consultado los últimos 100 tweets y se muestran los 10 conceptos que aparecen con más frecuencia.

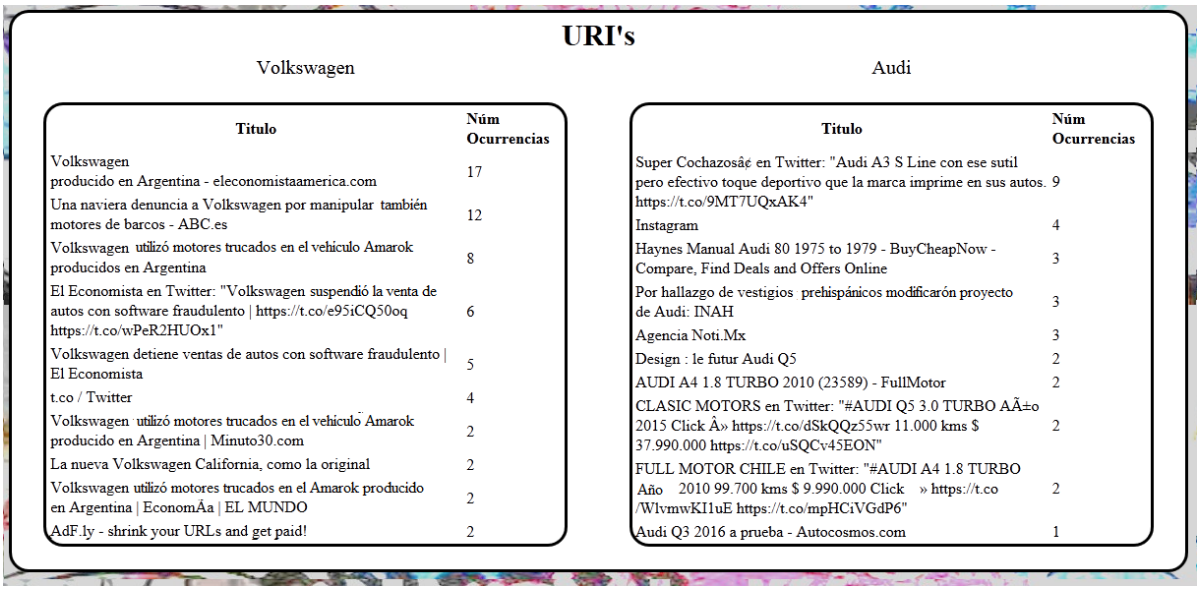
Figura 4. 13: Ejemplo de gráfico de conceptos

Cuantos menos resultados se muestren en el gráfico, más claro aparecerá este.

4.2.10. Pantalla de Resultados: Gráfico de URIs

En este caso mostramos una tabla con los títulos de las URIs que aparecen con más frecuencia en los tweets consultados.

Para este ejemplo se han consultado los últimos 100 tweets y se muestran los 10 títulos de URIs que aparecen con más frecuencia.



URI's

Volkswagen

Titulo	Núm Ocurrencias
Volkswagen producido en Argentina - eleconomistaamerica.com	17
Una naviera denuncia a Volkswagen por manipular también motores de barcos - ABC.es	12
Volkswagen utilizó motores trucados en el vehículo Amarok producidos en Argentina	8
El Economista en Twitter: "Volkswagen suspendió la venta de autos con software fraudulento | https://t.co/e95iCQ50oq https://t.co/wPeR2HUOx1"	6
Volkswagen detiene ventas de autos con software fraudulento | El Economista	5
t.co / Twitter	4
Volkswagen utilizó motores trucados en el vehículo Amarok producido en Argentina | Minuto30.com	2
La nueva Volkswagen California, como la original	2
Volkswagen utilizó motores trucados en el Amarok producido en Argentina | EconomÁa | EL MUNDO	2
AdF.ly - shrink your URLs and get paid!	2

Audi

Titulo	Núm Ocurrencias
Super Cochazosâ¢ en Twitter: "Audi A3 S Line con ese sutil pero efectivo toque deportivo que la marca imprime en sus autos. https://t.co/9MT7UQxAK4"	9
Instagram	4
Haynes Manual Audi 80 1975 to 1979 - BuyCheapNow - Compare, Find Deals and Offers Online	3
Por hallazgo de vestigios prehispánicos modificarón proyecto de Audi: INAH	3
Agencia Noti.Mx	3
Design : le futur Audi Q5	2
AUDI A4 1.8 TURBO 2010 (23589) - FullMotor	2
CLASIC MOTORS en Twitter: "#AUDI Q5 3.0 TURBO AÃ±o 2015 Click Â» https://t.co/dSkQQz55wr 11.000 kms $ 37.990.000 https://t.co/uSQCv45EON"	2
FULL MOTOR CHILE en Twitter: "#AUDI A4 1.8 TURBO Año 2010 99.700 kms $ 9.990.000 Click » https://t.co /WlvmwKI1uE https://t.co/mpHCiVGdP6"	2
Audi Q3 2016 a prueba - Autocosmos.com	1

Figura 4. 14: Ejemplo de gráfico de URIs

Cuantos menos resultados se muestren en el gráfico, más claro aparecerá este.

Capítulo 5

Implementación del sistema

En este capítulo de la memoria se especificará la implementación del sistema. Quedará especificada la estructura de ficheros de la aplicación y la arquitectura.

Anteriormente a la implementación de la aplicación se procedió a la instalación del software necesario para la fase de desarrollo y se dio de alta la aplicación en Twitter desde la URL https://dev.twitter.com/apps/new. A partir de que la aplicación ya es dada de alta se dispone de la consumer_key y la consumer_secret que identifican a la aplicación de manera única.

También es necesario disponer de una cuenta en MeaningCloud para el análisis del texto.

5.1. Arquitectura del sistema

Como se ha hablado en el capítulo 2 de este documento, para la elaboración del código fuente de esta aplicación se ha utilizado el framework Django, que sigue un patrón basado en MVC[11], el MTV[12].

La aplicación separa la lógica de negocios de la lógica de presentación usando un sistema de plantillas.

En el patrón de diseño MTV:

- M significa "Model" (Modelo), la capa de acceso a la base de datos. Esta capa contiene toda la información sobre los datos. Como acceder a estos, como validarlos, cual es el comportamiento que tiene, y las relaciones entre los datos.

- T significa "Template" (Plantilla), la capa de presentación. Esta capa contiene las decisiones relacionadas a la presentación, como algunas cosas son mostradas sobre una página web u otro tipo de documento.

- V significa "View" (Vista), la capa de la lógica de negocios. Esta capa contiene la lógica que accede al modelo y la delega a la plantilla apropiada. Se puede tomas como un puente entre los modelos y las plantillas.

[11] Modelo-Vista-Controlador
[12] Modelo-Vista-Template

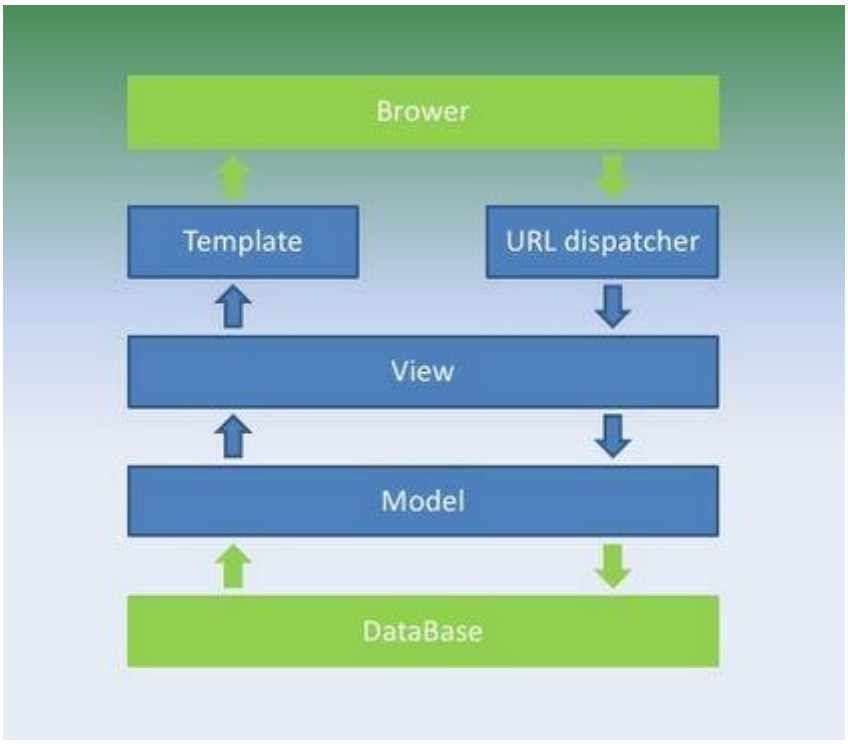


Figura 5. 1: Esquema del MTV

5.2. Diseño a bajo nivel

Al crear el proyecto en el directorio elegido con Django, éste nos genera los siguientes archivos y directorios.

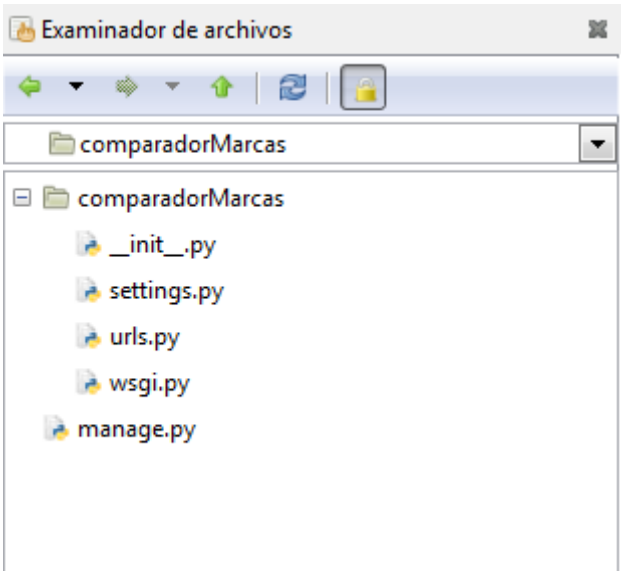


Figura 5. 2: Estructura de ficheros al crear el proyecto

A partir de generar nuestro proyecto, hay que crear la aplicación dentro del proyecto, lo que nos deja la siguiente estructura de directorios y archivos.

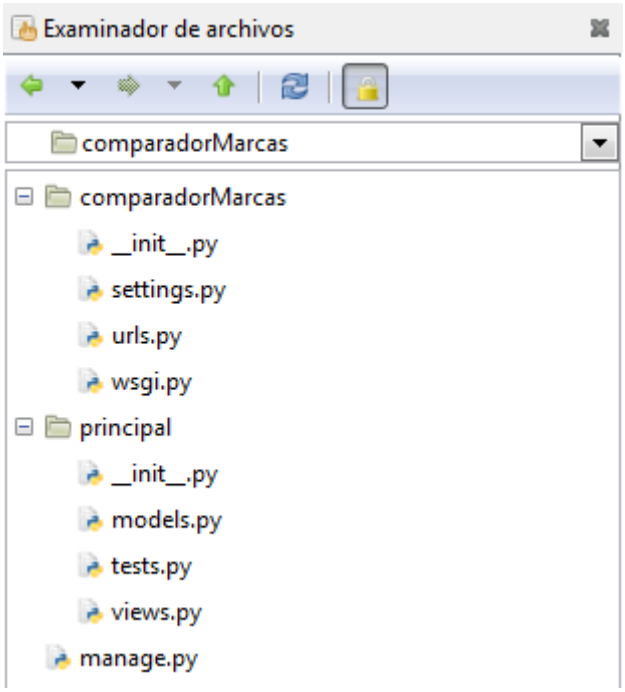


Figura 5. 3: Estructura de ficheros al crear la aplicación dentro del proyecto

A partir de que tenemos la aplicación creada pasamos a modificar los ficheros para adaptarlos a nuestro proyecto.

5.2.1. Fichero settings.py

Este archivo es el que permite configurar la conexión a la base de datos, la zona horaria, el idioma, los directorios principales del proyecto, las aplicaciones del proyecto (puede haber más de una aplicación por proyecto), entre otras cosas.

Codificación de caracteres

Nuestro idioma está lleno de caracteres especiales por lo tanto lo primero será agregar la siguiente línea de código al archivo `settings.py`:

```
#encoding:utf-8
```

Ruta del proyecto

Configurar la ruta del proyecto permitirá lanzar la aplicación desde cualquier directorio y mover el proyecto a cualquier PC con Django instalado. Para ello se escriben las siguientes líneas en el archivo `settings.py`:

```
# Build paths inside the project like this: os.path.join(BASE_DIR, ...)
import os

BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
RUTA_PROYECTO = os.path.dirname(os.path.realpath(__file__))
```

Figura 5. 4: Configuración de la ruta del proyecto en Setting.py

Zona horaria e idioma

Django permite configurar la zona horaria del proyecto (*Figura 5. 5: Configuración de Idioma y Horario en settings.py*).

Django también permite configurar el idioma predeterminado para el funcionamiento del proyecto (*Figura 5. 5: Configuración de Idioma y Horario en settings.py*)

```
# Internationalization
# https://docs.djangoproject.com/en/1.8/topics/i18n/

LANGUAGE_CODE = 'es-ES'

TIME_ZONE = 'Europe/Madrid'
```

Figura 5. 5: Configuración de Idioma y Horario en settings.py

Aplicaciones instaladas

Como hemos dicho anteriormente un proyecto con Django puede tener varias aplicaciones y es necesario configurarlas en el fichero `settings.py`. Aparte de las aplicaciones creadas, también hay que configurar la aplicación de administración y documentación, aunque al crear el proyecto estas ya vienen configuradas por defecto.

```
# Application definition

INSTALLED_APPS = (
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'principal',
)
```

Figura 5. 6: Configuración de Apps en settings.py

Directorio de plantillas

La aplicación dispone de un directorio de plantillas ubicado en el mismo directorio que el fichero `settings.py`. En este directorio de plantillas se incluyen los ficheros `.html` para el formulario de entrada y para el pintado de resultados (*5.2.10. Estructura final de la aplicación*).

Para que el proyecto funcione correctamente es necesario configurar la ruta del directorio de las plantillas en el fichero `settings.py` de la siguiente manera:

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(RUTA_PROYECTO,'plantillas'),],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.core.context_processors.media',
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
```

Figura 5. 7: Configuración de la ruta del directorio 'plantillas' en settings.py

Directorio de contenido estático

La aplicación dispone de un directorio estático llamado `media` donde se almacenan los ficheros estáticos necesarios para la aplicación. Este directorio se encuentra también a la altura del fichero `settings.py`.

Al igual que en el caso del directorio `plantillas`, también es necesario configurar la ruta en el fichero `settings.py` de la siguiente manera:

```
# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/1.8/howto/static-files/

STATIC_URL = '/static/'

MEDIA_ROOT = os.path.join(RUTA_PROYECTO,'media')

MEDIA_URL = '/site_media/'
```

Figura 5. 8: Configuración de la ruta del directorio 'media' en settings.py

Con estas modificaciones ya está el fichero `settings.py` listo.

5.2.2. Fichero urls.py

El fichero `urls.py` contiene las direcciones URL del proyecto en Django.

Es necesario modificar este fichero de la siguiente manera:

```
from django.conf.urls import include, url
from django.contrib import admin
from django.conf import settings

urlpatterns = [
    url(r'^$', 'principal.views.formularioComparar'),
    url(r'^admin/', include(admin.site.urls)),
    url(r'^site_media/(?P<path>.*)$','django.views.static.serve', {'document_root': settings.MEDIA_ROOT}),

]
```

Figura 5. 9: Configuración de las URLs en el fichero urls.py

Como se ve en la figura queda configurada la URL de entrada a la aplicación y la URL donde se encuentran los ficheros estáticos que la aplicación necesita.

En la figura de arriba se ve que la URL de entrada de la aplicación se forma pasando como parámetro un método del fichero `views.py` (*5.2.4. Vistas*).

5.2.3. Formularios

Nuestra aplicación posee un formulario de entrada con varios campos.

En el caso de esta aplicación se ha querido diferenciar cuatro formularios pero todos representados en una misma página de inicio. Los cuatro formularios son los siguientes.

- **Datos de Twitter**
  Los parámetros que se solicitan en este formulario son Access Token Key, Access Token Secret y el número de tweets que se quieren consultar
- **Datos de MeaningCloud**
  Este formulario consta de un único parámetro que es License Key.
- **Marcas**
  Se solicita como parámetros el nombre de las dos marcas que se quiere consultar.
- **Número de Resultados a mostrar**
  En este formulario hay que introducir el número de resultados que se quieren mostrar en la pantalla de resultados. Este parámetro es útil para los resultados del análisis de entidades, conceptos y URI`s. Los resultados pueden ser muy numerosos por eso se puede acotar la búsqueda al número que se desee.

Para definir los formularios en Django se crea un fichero llamado `forms.py` en la carpeta de la aplicación, que en este caso se llama `principal` (*5.2.10. Estructura final de la aplicación*).

La clase ha quedado de la siguiente forma.

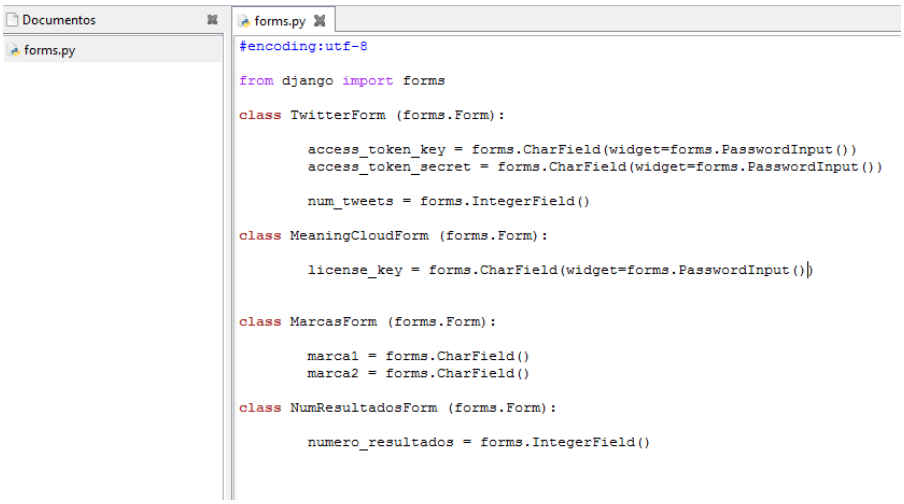


Figura 5. 10: Fichero forms.py

Como se muestra en la figura el fichero contiene cuatro formularios y dentro de cada uno de ellos están definidos los campos del formulario correspondientes.

Tenemos dos tipos de campos en los formularios, cajas de texto y cajas de números enteros.

5.2.4. Vistas

Un función de vista o una vista, es una función en Python que hace una solicitud Web y devuelve una respuesta Web, esta respuesta puede ser el contenido de una página, un error 404, una imagen, un documento XML, entre muchas cosas más.

La vista contiene toda la lógica necesaria para devolver una respuesta, todas estas respuestas se encuentran en un único archivo y este archivo se llama: `views.py`, que se encuentra dentro de cada aplicación de Django, en nuestro caso dentro del directorio llamado `principal` (*5.2.10. Estructura final de la aplicación*).

En nuestro fichero `views.py` tan solo tenemos una vista y es la que muestra la plantilla del formulario de entrada. Dentro de esta vista mostramos la plantilla (ver *5.2.5. Plantillas*) de inicio en la primera iteración o llamamos al método que inicia el procesado de los datos recibidos para la elaboración del análisis.

En la siguiente figura se ven los Imports, definición de la función `comparar` y recogida de los parámetros del formulario en `views.py`:

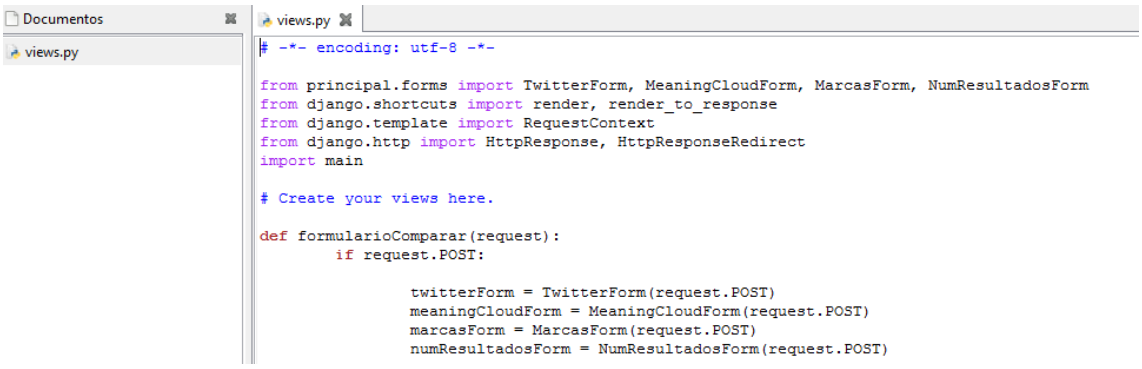


Figura 5. 11: Ejemplo de definición de funciones en views.py

Dentro de este método y cuando nos han enviado datos por el formulario recogemos los tweets, los analizamos y llamamos a la plantilla correspondiente para mostrar los resultados (*5.2.8. Análisis del texto*).

```
return render(request, 'resultadoComparar.html', {'marca1': marca1, 'marca2':marca2,
'clasesPrincipal1':clasesPrincipal1, 'clasesPrincipal2': clasesPrincipal2, 'clasesDetalle1': clasesDetalle1, 'clasesDetalle2':
clasesDetalle2, 'entidades1':entidades1, 'entidades2':entidades2, 'conceptos1': conceptos1, 'conceptos2': conceptos2, 'urisLabel1':
uris_label1, 'urisLabel2': uris_label2, 'urisOccr1': uris_occr1, 'urisOccr2': uris_occr2})
```

Figura 5. 12: Redirección a una plantilla desde el fichero views.py

En caso de que no haya parámetros que recoger desde los formularios, es decir, que sea la página de inicio la que queremos mostrar:

```
else:
twitterForm = TwitterForm()
meaningCloudForm = MeaningCloudForm()
marcasForm = MarcasForm()
numResultadosForm = NumResultadosForm()
return render(request, 'formularioComparar.html', {'twitterForm': twitterForm, 'meaningCloudForm': meaningCloudForm,
'marcasForm': marcasForm, 'numResultadosForm': numResultadosForm})
```

Figura 5. 13: Redirección a la pantalla de inicio desde views.py

5.2.5. Plantillas

Las plantillas, en nuestro caso dos páginas `html`, se encuentran en el directorio `plantillas` cuya ruta quedó definida en fichero `settings.py` (*5.2.10. Estructura final de la aplicación*).

En la plantilla de inicio es donde se pinta el formulario de recogida de parámetros de la siguiente manera:

```
<form id='formulario' method='post' enctype='multipart/form-data' action=''>{% csrf_token %}
        <fieldset id='primerForm'>
                <legend>Datos Twitter</legend>
                        {{twitterForm.as_p}}
        </fieldset>
        <fieldset>
                <legend>Datos Meaning Cloud</legend>
                        {{meaningCloudForm.as_p}}
        </fieldset>
        <fieldset>
                <legend>Marcas</legend>
                        {{marcasForm.as_p}}
        </fieldset>
        <fieldset>
                <legend>Número de Resultados a mostrar</legend>
                        {{numResultadosForm.as_p}}
        </fieldset>
        <fieldset id='boton'>
                <center><input type='submit' value='Comparar'/></center>
        </fieldset>
</form>
```

Figura 5. 14: Formulario en una plantilla html

En la plantilla donde se visualizan los resultados se pintan los gráficos que han sido generados durante la ejecución de la aplicación y almacenados en el directorio `media` (*5.2.6. Recursos estáticos*) (*5.2.10. Estructura final de la aplicación*).

Las plantillas de este proyecto incluyen un fichero `css` para definir el estilo de las páginas. El fichero `css` es un recurso estático por lo que se trata en el siguiente apartado.

5.2.6. Recursos estáticos

Tanto el contenido estático como las imágenes de los gráficos generados durante la ejecución del programa y el archivo `css` se encuentran en el directorio `media` (*5.2.10. Estructura final de la aplicación*).

La ruta de este directorio ya quedó definida en el fichero `settings.py`

Un ejemplo de cómo se incluye una imagen guardada en el directorio de recursos estáticos:

```
<img src='{{MEDIA_URL}}imagenes/sentimientos_marca1_{{ID}}.jpg'
alt='sentimientos' id='grafico'>
```

El `css` se incluye en las plantillas `html` de la siguiente manera:

```
<link rel="stylesheet" href="{{MEDIA_URL}}css/estilo.css">
```

Los ficheros `Java script` también pueden almacenarse en este directorio. En el caso de este proyecto, al tratarse de funciones muy pequeñas ha quedado embebido en la plantilla `html`.

5.2.7. Llamada a Twitter

La llamada a Twitter se hace desde un directorio llamado `llamadaTwitter.py` y este fichero se encuentra en el directorio de la aplicación (*5.2.10. Estructura final de la aplicación*).

En este fichero es necesario importar la librería `TwitterAPI` de Python para realizar la consulta a la API de Twitter.

También en este fichero, se guarda el valor de la consumer-key y consumer-secret de la app creada en Twitter.

El fichero se muestra a continuación.

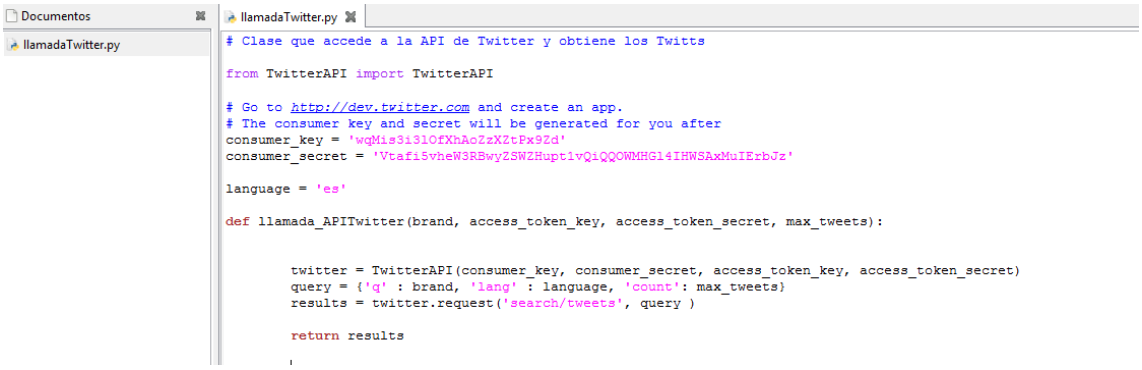


Figura 5. 15: Fichero donde se llama a la API de Twitter

5.2.8. Análisis del texto

Una vez que se ha hecho la llamada a Twitter y se han obtenido los tweets es hora de analizar el texto.

Para el análisis del texto se han utilizado tres APIs diferentes de MeaningCloud.

Sentiment Analysis

Con esta API se consulta el sentimiento del tweet.

La llamada a esta API se encuentra en un fichero llamado `analizaSentiment.py` en el directorio de la aplicación (*5.2.10. Estructura final de la aplicación*).

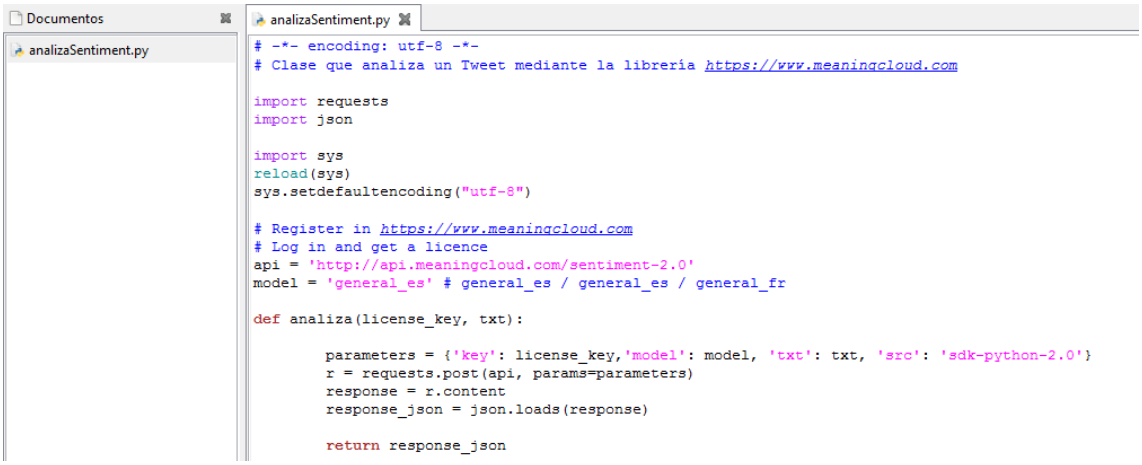


Figura 5. 16: Fichero donde se llama a la API Sentiment Analysis de MeaningCloud

Text Classification

Con esta API se clasifica el tweets en diferentes temas o clases.

Existe una clasificación de temas principal que coincide con el primer nivel de clasificación de IPTC y una clasificación más detallada que coincide con el segundo nivel de IPTC.

La llamada a esta API se encuentra en un fichero llamado `analizaClass.py` en el directorio de la aplicación (*5.2.10. Estructura final de la aplicación*).

```
# -*- encoding: utf-8 -*-

import json
import requests

# We define the variables need to call the API
api = 'http://api.meaningcloud.com/class-1.1'
model = 'IPTC_es' #IPTC_es/IPTC_en/IPTC_fr/IPTC_it/IPTC_ca/EUROVOC_es_ca/BusinessRep_es/BusinessRepShort_es

def analiza(key, txt):
        # We make the request and parse the response
        parameters = {'key': key, 'model': model, 'txt': txt, 'src': 'sdk-python-1.1'}
        r = requests.post(api, params=parameters)
        response = r.content
        response_json = json.loads(response)

        if response_json['status']['code'] == '0':

                category_list_label = []
                category_list_label_1 = []
                category_list_label_2 = []

                try:
                        categories = response_json['category_list']

                        if len(categories)>0:

                                for index in range(len(categories)):

                                        category_list_label = categories[index]['label'].split("-")

                                        category_list_label_1.append(category_list_label[0])
                                        category_list_label_2.append(category_list_label[1])


                except KeyError:
                        category_list_label_1 = category_list_label_1
                        category_list_label_2 = category_list_label_2
                mensaje = "Error al Analizar el tweet con MeaningClould: " + response_json['status']['code'] + " - " + response_json
['status']['msg']
                return '!OK', mensaje
```

Figura 5. 17: Fichero donde se llama a la API Text Classification de MeaningCloud

Topics Extraction

Con esta API se obtienen las entidades, conceptos y URIs que contienen los tweets seleccionados.

```
# -*- encoding: utf-8 -*-

import requests
import json

# We define the variables need to call the API
api = 'http://api.meaningcloud.com/topics-1.2'
lang = 'es' #es/en/fr/it/pt/ca

def analiza(key, txt):

        # We make the request and parse the response
        parameters = {'key': key,'lang': lang, 'txt': txt, 'tt': 'a', 'src': 'sdk-python-1.2'}
        r = requests.post(api, params=parameters)
        response = r.content
        response_json = json.loads(response)

        if response_json['status']['code'] == '0':

                entidades = []
                paises = []
                conceptos = []
                uris = []

                try:
                  if len(response_json['entity_list']) > 0:
                    entities = response_json['entity_list']
                    info_type = ''
                    for index in range(len(entities)):
                      entidades = entidades + [entities[index]['form']]
                      try:
                        paises = paises + [entities[index]['semgeo_list'][0]['country']['form']]
                      except KeyError:
                        pass
                  else:
                    entidades = entidades
                    paises = paises
                except KeyError:
                   entidades = entidades
                   paises = paises

                try:
                  if len(response_json['concept_list']) > 0:
                    concepts = response_json['concept_list']
                    info_type = ''
                    for index in range(len(concepts)):
                      conceptos = conceptos + [concepts[index]['form']]
                  else:
                    conceptos = conceptos
                except KeyError:
                  conceptos = conceptos

                try:
                  if len(response_json['uri_list']) > 0:
                    uriss = response_json['uri_list']
                    for index in range(len(uriss)):
                      uris = uris + [uriss[index]['form']]
                  else:
                    uris = uris
                except KeyError:
                  uris = uris

                return entidades, paises, conceptos, uris

        else:
                mensaje = "Error al Analizar el tweet con MeaningClould: " + response_json['status']['code'] + " - " + response_json
['status']['msg']
                return '!OK', mensaje
```

Figura 5. 18: Fichero donde se llama a la API Topic Extraction de MeaningCloud

La llamada a esta API se encuentra en un fichero llamado `analizaTopic.py` en el directorio de la aplicación (*5.2.10. Estructura final de la aplicación*).

Estas tres librerías devuelven un objeto JSON que contiene toda la información extraída del análisis del texto de cada uno de los tweets, por tanto, estas tres clases se encargan de la extracción de la información del objeto JSON y su clasificación para más tarde representar los resultados.

5.2.9. Representación de resultados

Una vez que hemos analizado el texto de los tweets procedemos a pintar los resultados obtenidos en gráficos. Para ellos llamamos a Matplotlib y guardamos las imágenes de los gráficos generadas en nuestro directorio de recursos estáticos (*5.2.10. Estructura final de la aplicación*).

Las imágenes que se generan se guardan con el siguiente patrón: *nombreGrafico+ marca + ID.*

Periódicamente se borran las imágenes de este directorio.

Se dibujan tres tipos de gráficos: grafico de sectores, grafico de barras y gráficos de texto. Y una tabla.

Ejemplo de gráfico de sectores:

```
plt.pie(sizes, labels=labels, colors=colors, autopct='%1.1f%%',
shadow=True)
```

Ejemplo de gráfico de barras:

```
plt.barh(index, sizes, align="center")
```

Ejemplo de gráfico de texto:

```
plt.text(x, y, eq, ha='center', va='center', color=color,
alpha=alpha, transform=plt.gca().transAxes, fontsize=size,
clip_on=True)
```

En el gráfico de texto, cada uno de los textos se representa de manera aleatoria dentro del gráfico y los textos que aparezcan con mayor frecuencia estarán representados con el tamaño de letra superior. En estos gráficos además aparece una leyenda con el texto y el número de veces que aparece.

5.2.10. Estructura final de la aplicación

Después de la creación inicial del proyecto y de las clases que hemos añadido hasta completar nuestra aplicación, la estructura ha quedado de la siguiente manera:

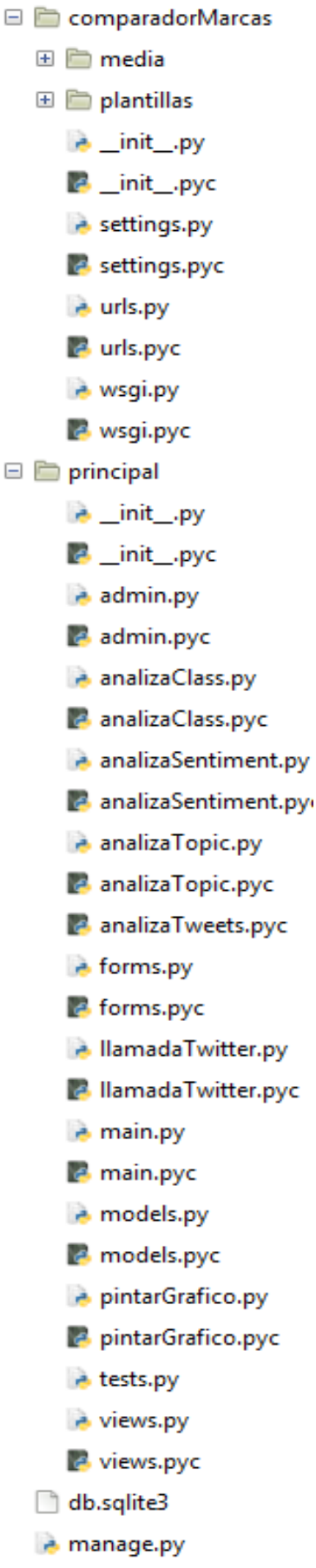


Figura 5. 19: Estructura final del proyecto

En el directorio `media` es donde se encuentran los recursos estáticos y queda de la siguiente manera:

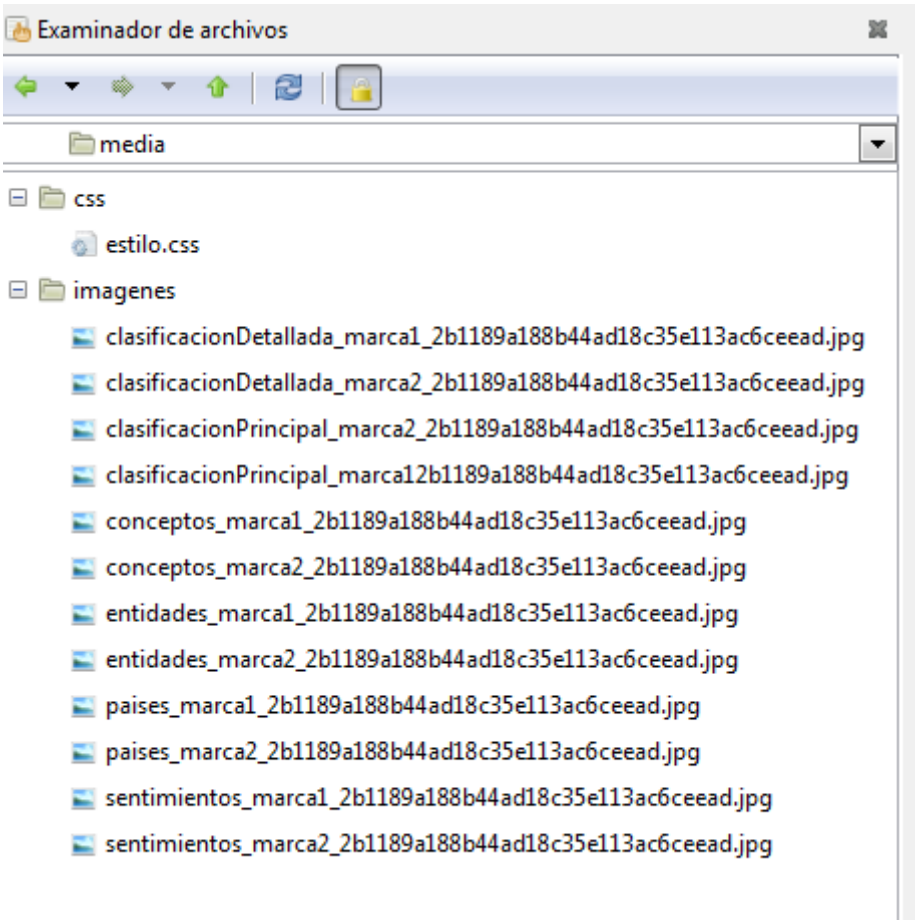


Figura 5. 20: Estructura final del directorio media

En el directorio `plantillas` se encuentran las páginas `html` del proyecto:

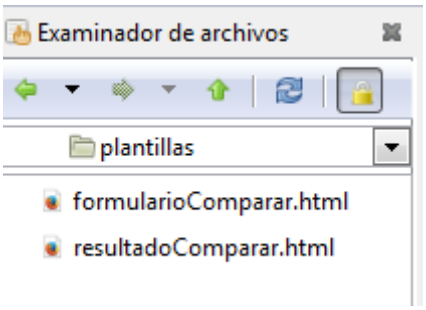


Figura 5. 21: Estructura final del directorio plantillas

Capítulo 6

Pruebas

Después de finalizar el desarrollo de la aplicación, esta fue sometida a un conjunto de pruebas funcionales para ver que el comportamiento de la aplicación era el correcto.

La aplicación debe cumplir con todos los requisitos que se especifican en el capítulo 3 de este documento (*3. Requisitos de diseño*).

A continuación se describen las pruebas realizadas para comprobar el correcto funcionamiento de este proyecto.

6.1. Usuario intenta hacer una consulta

Desde la primera pantalla el usuario intenta acceder a la comparación de dos marcas, se han comprobado los siguientes casos:

1. El usuario no consigue acceder a la consulta y se le notifica un mensaje de error en los siguientes casos:

    - El usuario no ha rellenado todos los campos del formulario.
    - Si el campo de número de tweets no es un número entero positivo.
    - Si el campo de número de resultados no es un número entero positivo.

2. El usuario consigue acceder a la consulta en los siguientes casos:

    - El usuario ha rellenado todos los campos del formulario.
    - El campo de número de tweets es un número entero positivo.
    - El campo de número de resultados es un número entero positivo.

6.2. Consulta a Twitter

Cuando el usuario ha introducido todos los parámetros en el formulario y se inicia la consulta a la API de Twitter, se han comprobado los siguientes casos:

1. La consulta falla y se muestra un mensaje de error al usuario en los siguientes casos:

    - El usuario no ha podido ser autenticado.
    - El token de acceso ha expirado o es incorrecto.

- Twitter está temporalmente fuera de servicio.
- Si se ha producido un error interno en Twitter.

2. La consulta ha ido correctamente, entonces el programa sigue su proceso.

No todos los casos de posible error han podido ser probados, pero al hacer la consulta a la API se recoge el código que devuelve y si es un código de error se muestra por pantalla al usuario junto con la descripción del error. Por tanto todos los posibles errores que puede devolver Twitter están contemplados.

6.3. Consulta a las APIs de MeaningCloud: Sentiment Analysis, Text Classification y Topics Extraction

Cuando se han obtenido los tweets consultando a través de la API de Twitter y se pasa al análisis de los tweets con las APIs de MeaningCloud, se han comprobado los siguientes casos:

1. El análisis de los tweets falla y se muestra un mensaje de error a los usuarios en los siguientes casos:

    - Si la operación ha sido denegada.
    - Si han sido superados los créditos por suscripción
    - Si el número de parámetros en la consulta no es correcto.
    - Si se ha producido un error interno.
    - Si no se puede conectar con el servicio.

2. El análisis ha ido correctamente y el programa sigue su ejecución.

Al igual que en la consulta a la API de Twitter no ha sido posible probar todos los casos de error, pero al hacer la consulta a las APIS de MaeaningCloud se recoge el código que devuelve la consulta y si es un código de error se muestra por pantalla al usuario junto con la descripción del error. Así que todos los posibles errores que se puede producir al consultar las APIs de MeaningCloud quedan contemplados.

6.4. Presentación de los resultados

A la hora de pintar los gráficos del resultado de los análisis se han comprobado los siguientes casos:

1. El gráfico no se guarda en el directorio de recursos estáticos.

En este caso en el lugar del gráfico aparece un texto alternativo con el nombre del gráfico que debía mostrarse en ese lugar.

2. Al analizar el tweet no hay resultados para entidades, topics o URIs.

   En este caso y al ser gráficos de texto, el gráfico se pinta vacío, y la leyenda aparece vacía

3. Al analizar el tweet no hay resultados para el gráfico de clases.

   En este caso al ser un gráfico de barras, no aparece ninguna barra y la leyenda aparece vacía.

Capítulo 7

Conclusiones y trabajos futuros

En este capítulo se exponen todas las conclusiones obtenidas durante las fases de desarrollo del proyecto y se proponen posibles mejoras que se podrían añadir a la aplicación en un futuro.

7.1. Conclusiones

Como paso previo al desarrollo de la aplicación fue necesario hacer una investigación exhaustiva sobre las tecnologías elegidas para realizar el proyecto, tanto del lenguaje de programación Python, como del framework Django.

Con el desarrollo de este proyecto se ha buscado conocer nuevas tecnologías aparte de cumplir los requisitos especificados al principio. Fue muy importante el aprendizaje de un nuevo lenguaje y framework con el trabajo que eso conlleva, aparte de conocer una nueva herramienta como Matplotlib para la representación de datos mediante gráficos.

Este proyecto también aporta conocimiento sobre una de las redes sociales con más auge en el momento en el que nos encontramos. Se obtienen conocimientos sobre la API REST de Twitter gracias al uso de algunos métodos para realizar consultas.

También se aprende como analizar un texto a partir de las APIs de MeaningCloud, ejecutando sus métodos. Un problema que se encontró fue que se superó el número de consultas con el plan gratuito por lo que se tuvo que aumentar el límite de dichas consultas.

Después de todo se puede confirmar que esta aplicación cumple con todos los requisitos especificados al inicio del desarrollo y además ha aportado nuevos conocimientos sobre distintas tecnologías.

7.1.1. Análisis comparativo de marcas

La aplicación puede ser de gran utilidad para conocer la opinión real de los usuarios con respecto a un producto o marca. Una empresa determinada podría hacer una valoración de los datos obtenidos por la aplicación y ver que posibles aspectos podría mejorar para contentar al usuario, en definitiva a partir de los resultados podría elaborar un plan a seguir para mejorar los aspectos que más puedan descontentar a los usuarios.

Con la comparativa de marcas que hace esta aplicación una empresa es capaz de conocer las fortalezas y debilidades de su principal competidor y compararlas con las

suyas propias para así elaborar una buena estrategia para el futuro y destacar por encima de sus principales competidores.

7.2. Trabajos futuros

La aplicación desarrollada cumple con los requisitos definidos pero es una aplicación a la que se podrían añadir algunas mejoras:

Mejorar el rendimiento

Uno de los mayores problemas de la aplicación es el tema del rendimiento a la hora de analizar el texto de cada tweet. Es un proceso muy laborioso por lo que el tiempo que tarda es bastante grande. En la actualidad los tweets son analizados en serie por lo que el tiempo de respuesta es bastante alto. Como posible mejora existe la opción de ir analizando los textos en paralelo, de 10 en 10 por ejemplo. Así conseguiríamos reducir el tiempo de respuesta.

Acceso por usuario

Actualmente se solicita como parámetro de entrada el access token secret y access token key así que otra de las mejoras posible es solicitar por parámetro el usuario y contraseña de Twitter y a través de las APIs de Twitter obtener el access token secret y access token key correspondientes al usuario. Siempre será más fácil para el usuario recordar su nombre de usuario y contraseña.

Comparación N marcas

En la actualidad sólo es posible consultar dos marcas, pero en un futuro podría ampliarse la consulta a N marcas en paralelo.

Exportación de resultados

Una posible mejora de la aplicación es poder exportar los resultados obtenidos tras el análisis del texto, por ejemplo a un fichero Excell.

Añadir otros tipos de análisis

Se podrían añadir más tipos de análisis aparte de los que ya se realizan.

Clasificación propia de modelos

Actualmente la clasificación es según las clasificaciones de MeaningCloud pero sería posible realizar una clasificación propia.

Análisis del sentimiento orientado a concepto

Actualmente se analiza el sentimiento de manera global pero en un futuro podría analizarse el sentimiento también a nivel de un único concepto.

Apéndice A

Planificación

En este apéndice se presenta la planificación del proyecto y los costes que ha conllevado su realización.

Para la planificación se ha dividido el proyecto en un conjunto de tareas, y cada una de estas tareas tiene asociado su tiempo de elaboración.

Para el cálculo del presupuesto se ha tenido en cuenta la duración total del proyecto y el sueldo estimado para un Ingeniero Técnico de Telecomunicaciones.

El presupuesto calculado es el resultado de la suma de los costes de personal que ha participado en el desarrollo del proyecto, los costes de los materiales empleados y un 20 % adicional del presupuesto de costes indirectos, como electricidad, Internet, etc.

A.1. Distribución temporal

Las distintas tareas en las que ha quedado dividido el proyecto y el tiempo que ha llevado su elaboración son las siguientes:

1. Documentación e investigación (~ 15 días).

    - Búsqueda de información sobre el tipo de aplicación a desarrollar.
    - Búsqueda de información sobre las tecnologías escogidas para la realización del proyecto.

2. Instalación del équido de desarrollo (~ 3 días).

    - Instalación del lenguaje de programación
    - Instalación del framework escogido.
    - Instalación de Matplotlib.
    - Creación de cuenta de Twitter.
    - Creación de cuenta en MeaningCloud.

3. Análisis de requisitos (~ 5 días).

    - Enumeración y descripción de la lista de requisitos.

4. Diseño del sistema (~ 5 días).

    - Diseño de la estructura de la aplicación.

- Diseño de la interfaz.

5. Implementación (~ 40 días).

    - Implementación de las clases.
    - Implementación de las plantillas.

6. Pruebas (~ 5 días).

    - Pruebas para el correcto funcionamiento de la aplicación.

7. Redacción de la memoria (~ 20 días).

    - Redacción del presente documento.

A.2. Presupuesto

Para la elaboración del presupuesto de este proyecto se han tenido en cuenta los siguientes datos:

- Sueldo base que se considera para un Ingeniero Técnico en Telecomunicaciones: 50 euros/hora.

- Las jornadas se consideran de 8 horas al día.

A.2.1. Costes de personal

Los costes asociados al personal se han calculado en función al salario base para un Ingeniero Técnico de Telecomunicación y al número de horas trabajadas.

El cálculo queda de la siguiente manera.

Tarea	**Horas de trabajo**
Documentación e investigación	120
Instalación del équido de desarrollo	24
Análisis de requisitos	40
Diseño del sistema	40
Implementación	320
Pruebas	40
Redacción de la memoria	160
Coste (€)	**50 x 744 = 37200**

Tabla C. 1: Costes de personal

A.2.2. Costes de materiales

En los costes materiales únicamente hemos tenido en cuenta el ordenador de desarrollo de la aplicación.

Concepto	Coste (€)	% Uso dedicado proyecto	Dedicación (meses)	Periodo de depreciación	Coste imputable (€)
Portátil Intel Core	**600**	**100**	**4**	**60**	**40**

Tabla C. 2: Costes de materiales

Para calcular este coste se ha empleado la siguiente fórmula de amortización:

$$\frac{A}{B} \times C \times D$$

Donde:

A = Número de meses desde la fecha de facturación en que el equipo es utilizado

B = Período de depreciación

C = Coste del equipo (sin IVA)

D = % del uso que se dedica al proyecto

A.2.3. Costes totales

Los costes totales del proyecto se calculan mediante la suma de costes de personal, costes de material y los costes indirectos de un 20%. En estos costes indirectos se incluyen electricidad, Internet, etc.

Concepto	Importe (€)
Costes de personal	37200
Costes de materiales	40
Costes indirectos (20%)	7448
Total	**(37200+40)x1.20 =44688**

Tabla C. 3: Costes totales

Bibliografía

[1] «Wikipedia, artículo Twitter,» [En línea]. Available: http://es.wikipedia.org/wiki/Twitter. [Último acceso: Septiembre 2015].

[2] «Wikipedia, artículo Python,» [En línea]. Available: https://es.wikipedia.org/wiki/Python. [Último acceso: Septiembre 2015].

[3] «Página de Python Software Foundation,» [En línea]. Available: https://www.python.org/. [Último acceso: Septiembre 2015].

[4] «Mundo geek. Mi primer programa en Python,» [En línea]. Available: http://mundogeek.net/archivos/2008/01/16/mi-primer-programa-en-python/. [Último acceso: Septiembre 2015].

[5] «Wikipedia, artículo Django (framework),» [En línea]. Available: https://es.wikipedia.org/wiki/Django_%28framework%29#Requerimientos. [Último acceso: Septiembre 2015].

[6] «Django | Guía rápida de instalación,» [En línea]. Available: http://django.es/docs/intro/instalacion/. [Último acceso: Septiembre 2015].

[7] «Descarga Django | Django en Español, django-eses,» [En línea]. Available: http://django.es/descarga/. [Último acceso: Septiembre 2015].

[8] «Tutorial Matplotlib,» [En línea]. Available: http://es.slideshare.net/diegocamilopenaramirez5/tutorial-de-matplot-lib. [Último acceso: Septiembre 2015].

[9] «Python Cientifico. Introducción a Numpy y Matplotlib,» [En línea]. Available: http://es.slideshare.net/kikocorreoso/python-cientfico-introduccin-a-numpy-y-matplotlib. [Último acceso: Septiembre 2015].

[10] «Descargar e instalar Matplotlib,» [En línea]. Available: http://blog.espol.edu.ec/icm00794/descargar-e-instalar-matplotlib/. [Último acceso: Septiembre 2015].

[11] «Twitter, API REST,» [En línea]. Available: https://dev.twitter.com/docs/api. [Último acceso: Septiembre 2015].

[12] «OAuth Core 1.0,» [En línea]. Available: http://oauth.net/core/1.0/#anchor9. [Último acceso: Septiembre 2015].

[13] «Return (Gis), artículo Autenticación OAuth,» [En línea]. Available: http://returngis.net/2010/05/. [Último acceso: Septiembre 2015].

[14] «Be code my friend, artículo Entendiendo OAuth con Twitter,» [En línea]. Available: http://www.becodemyfriend.com/2010/10/entendiendo-oauth-con-twitter-from-scratch/. [Último acceso: Septiembre 2015].

[15] «MeaningCloud,» [En línea]. Available: https://www.meaningcloud.com/es/. [Último acceso: Septiembre 2015].

[16] «Sngular Meaning,» [En línea]. Available: http://www.sngularmeaning.team/71/. [Último acceso: Septiembre 2015].

[17] «Wikipedia, artículo CSS,» [En línea]. Available: http://es.wikipedia.org/wiki/Hojas_de_estilo_en_cascada. [Último acceso: Septiembre 2015].

[18] «Guía breve de CSS,» [En línea]. Available: http://www.w3c.es/Divulgacion/GuiasBreves/HojasEstilo. [Último acceso: Septiembre 2015].

[19] «Wikipedia, artículo Estudio de mercado,» [En línea]. Available: https://es.wikipedia.org/wiki/Estudio_de_mercado. [Último acceso: Septiembre 2015].

[20] «Herramientas online para estudios de mercado,» [En línea]. Available: Herramientas online para estudios de mercado. [Último acceso: Septiembre 2015].